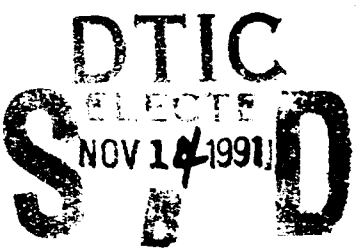
Technical Report 1450
October 1991

Parallel Processing Applied to Computational Electromagnetics

L. C. Russell
J. W. Rockway



NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152–5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed under project RV36I21 of the Computer Technology Block Program (N02D). This block is managed by the Independent Exploratory Development (IED) Program of the Naval Ocean Systems Center, under the guidance and direction of the Office of the Chief of Naval Research (OCNR–10P). This work was funded under program element 0602936N and was performed in FY 1991 by Linda C. Russell and John W. Rockway of Codes 824 and 805 respectively, Naval Ocean Systems Center, San Diego, CA 92152–5000.

Released by
J. B. Rhode, Head
EM Technology & Systems
Branch

Under authority of
R. J. Kochanski, Head
Shipboard Communications
Division

MA

SUMMARY

OBJECTIVE

To apply parallel processing techniques to a computational electromagnetic code and show that high-performance computing can improve the utility and efficiency of techniques used in ship electromagnetic designs.

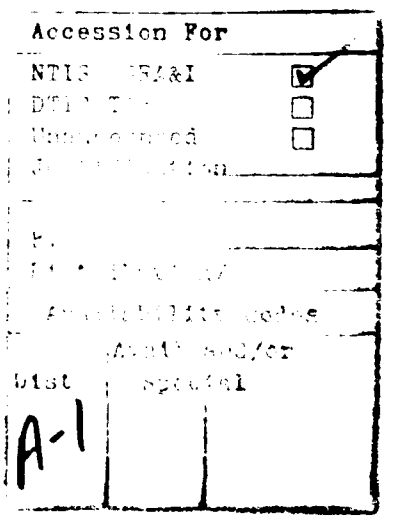
RESULTS

- Efficient parallel matrix filling, factoring, and solving routines were developed, implemented, and evaluated.
- The performances of the parallel matrix factoring and solving routines were independent of the structure being analyzed, whereas the performance of the parallel matrix filling routine was found to be very dependent on the structure being analyzed.
- Several different techniques for mapping the matrix columns onto the processors were implemented and evaluated.
- Efficiencies for all parallel algorithms increased as the size of the matrix being solved increased.
- Performance comparisons were made between a MicroVax, the four transputer array, and the Convex Model C-220 mini-supercomputer.

PAYOFFS

Several payoffs have resulted, or will result, from the work done on this IED project:

- A computer platform now exists for the efficient transition of computational electromagnetics to high-performance computing. The transputer array can be used to initiate parallelizing an existing computer code.
- This next-generation increase in processing speed will permit more accurate modeling of ship topsides and extend the practical method of moments modeling of ships to high HF through low UHF bands. This will make new, innovative ship designs much more feasible.
- Since topside synthesis is accomplished by iterative analysis, design quality will significantly improve with the enhanced speed offered by high-performance computing.



CONTENTS

FIGURES

TABLES

1.0 INTRODUCTION

This report presents the findings developed as a result of a Navy-funded Independent Exploratory Development (IED) investigation. This investigation applied parallel processing techniques to an existing computational electromagnetic (CEM) code. The goal was to demonstrate that high-performance computing (HPC) can improve the utility and efficiency of computational techniques used in ship electromagnetic designs. The parallel processing platform used was an array of four transputers on a board inside an IBM-compatible PC. The transputers were run using the ParaSoft Express operating environment. This operating environment was chosen to allow portability of the code to other HPC platforms and minimize the number of required programming changes to the CEM code. The CEM code was the method of moments (MoM) algorithm developed by the University of Houston.

Chapter 2 is a discussion of what parallel processing is and what it means in the world of HPC. Chapter 3 defines CEM, outlines the MoM algorithm to be parallelized, and describes why a high-speed computational capability is so important to CEM. Chapter 4 contains a description and justification of the approach that was used to parallelize the MoM algorithm. Chapter 5 describes the implementation of the sequential version of the code first on a PC and then on a single transputer. Chapters 6 and 7 are detailed descriptions of the technical aspects of parallelizing the different parts of the MoM code. Chapter 8 gives results and conclusions. Finally, chapter 9 discusses follow-on work and future technologies.

1.1 BENEFITS FROM IED PROJECT

Prior to presenting the details of this investigation, it is appropriate to comment on the overall benefits derived from the Navy IED program that funded this investigation of parallel processing applied to computational electromagnetics.

The funding of this IED task has permitted

- NOSC Principal Investigator (PI), Linda C. Russell, and Associate Investigator (AI), John W. Rockway, to establish contact with both the NOSC and the international HPC communities.
- Scientists and Engineers at the NOSC Model Range to become aware of parallel processing and HPC.
- The development of future CEM algorithms to be steered by HPC capabilities.
- Envisioning significant improvements in the design quality of the next-generation Navy ships.

2.0 PARALLEL PROCESSING

Traditional computers were designed around the von Neumann architecture. In this architecture, a single processor is connected to a single memory bank by a single communication bus. These are the computers with which most people are familiar. Vast improvements in processing capability were achieved by improvements in processor and memory technology. Most of these improvements were due to decreases in component sizes.

In the last decade, it became apparent that future improvements in processor speeds would not be as easy to achieve as they were in the past. Super high-speed computers (supercomputers) were built, but they were extremely costly to develop and usually difficult to maintain. They were reserved for only the world's most computational intensive problems. The average person did not have access to them.

Alternative architectures began to emerge. These architectures were based on the idea of achieving improved computational performance using existing processor technology. There are two fundamental ways of achieving this, pipelining and replicating. In pipelining, only one processor is used, but computational improvements are achieved by overlapping simple operations in time such that the processor is idle a very small percent of the time. Computations are overlapped with memory fetches and puts. The drawbacks to pipelining are that there are delays in setting up the pipeline and only simple repetitive operations can be pipelined. In contrast, replicating involves the use of more than one processor working simultaneously. Replicating is known as parallel (concurrent) processing.

2.1 CONCEPT OF PARALLEL PROCESSING

The concept of parallel processing had been around for a long time but only recently was the utility of it generally recognized. By 1986, more than a dozen companies were either selling or in the process of building parallel processors (Tazelaar, 1988).

The building blocks for parallel processor machines are the individual processors. These processors do not have to be very sophisticated - the computational speedup comes from the number of processors used. Often used as building blocks are the Intel 386, the Intel i860, or the INMOS transputer. These processors are connected to each other by high-speed communication links Usually each processor can be directly connected to no more than four other processors. This requires that the processors be configured into a communication architecture. Typical processor configurations include the hypercube, torus, binary tree, linear, and lattice mesh. The optimum configuration is a function of the number of processors, communication speeds, and the types of problems to be solved. Some parallel processing computers have software reconfigurable crossbar switches to allow simplified modification of the configuration.

The "beauty" of parallel processing is that it is a very inexpensive way to become familiar with the world of high-performance computing. A starter system of four transputers on a board which fits into a PC, along with the software needed to operate it, costs only about $5k. More processors can be added as one's budget allows. An actual parallel desktop supercomputer, Cogent Research's XTM, can be obtained for as low as $20k.

2.2 CLASSIFICATIONS OF PARALLEL COMPUTERS

The hardware used for parallel computers is classified in several ways. The main classification is between single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD). A parallel computer can also be either a shared memory or distributed memory system. This report is heavily oriented toward distributed memory MIMD systems, the classification of the transputer.

2.2.1 SIMD vs. MIMD

Traditional single processor von Neumann computers can be thought of as single instruction single data (SISD). This means that at any given moment only a single instruction can be operating on only a single data element. In a multiple processor SIMD system, there is still only a single instruction, but this time it is operating on multiple data elements in a lock-step fashion. This is similar to how vector processors work. In a multiple processor MIMD system, there are multiple instructions operating independently on multiple data elements. Today, many parallel computers are MIMD systems since this configuration offers the most flexibility. A MIMD system can emulate a SIMD system, but not the other way around. A SIMD system is, in general, less complicated than a MIMD system and, therefore, can be faster and less expensive. For certain problems, a SIMD system is well-suited and significant performance can be obtained. Other problems require a MIMD system.

2.2.2 Shared vs. Distributed Memory

In a shared-memory computer, there is only one bus to connect the processors to the memory. All processors are directly connected to the entire memory. The advantage is that every processor can see the entire memory. This saves execution time by reducing the amount of required interprocessor communication. The disadvantage is that a bottleneck can quickly develop on the bus if there are more than just a few processors . This is referred to as a von Neumann bottleneck.

In a distributed-memory computer, each processor has its own independent memory bank. A processor plus its memory is referred to as a node. Bus bottlenecks cannot occur. However, the system is now complicated by requiring the individual nodes to be able to communicate to each other. Communication delays can decrease system performance. Nonetheless, distributed memory is now the preferred configuration for today's parallel computers. For efficiency, it is a necessary requirement for massively parallel computers.

2.3 PARALLEL PROGRAMMING LANGUAGES AND OPERATING ENVIRONMENTS

In response to the proliferation of parallel hardware systems, a number of high-level languages with parallel extensions have been developed. These include parallel versions of Fortran, Pascal, Modula 2, and C (Davidson,1990). There are also some processor-specific languages such as Occam for the transputer. Processor-specific languages offer the best performance but also the least portability and require completely rewriting existing codes.

To facilitate program development on parallel computers, a number of parallel operating environments have been developed. Parallel operating environments allow the actual physical configuration of the hardware to become transparent to the user. The user does not have to know whether the processors are connected in a hypercube or lattice mesh configuration, or even how many processors there are. This allows the software to be much more portable between computers. Most parallel operating environments also offer performance-monitoring tools and parallel debuggers. Performance-monitoring and debugging are much more difficult for parallel programs than for sequential programs since not only can the operations and data be incorrect, but the timing may be off as well.

Express by the ParaSoft Corporation is a parallel operating environment designed to run on a number of different parallel machines (ParaSoft, 1990a). Express runs under the hosts standard operating system, can be used with both C and Fortran codes, supports dynamic load balancing, provides semiautomatic decomposition tools, and includes a source-level debugger and performance monitor. Code developed under Express has been stated to be portable between parallel machines provided Express is on both machines.

2.4 PARALLEL ALGORITHMS

Before deciding whether or not to use a parallel computer to solve a problem, the user must identify the inherent parallel aspects of the problem and determine a suitable hardware. Some problems have no inherent parallel aspects and should be run on a standard sequential computer. Many other problems have easily identifiable parallel aspects. The effectiveness of parallel computing depends on how well one can identify the inherent parallelism in a problem, develop an algorithm, and map it onto a suitable architecture. The optimal parallel algorithm is not necessarily the optimal serial algorithm.

There are two basic methods to parallelize a program on a MIMD computer: domain decomposition and algorithmic decomposition. Domain decomposition consists of having each processor run the same code, but with the data distributed among the processors. As an example, for a matrix problem the columns or rows of the matrix may be distributed among the processors in some fashion. Algorithmic decomposition consists of dividing a program into sections, and putting different sections of the code on the different processors. The different sections must be able to run independently. The bulk of the work involved in parallelizing any code is in determining the correct mix of domain decomposition and algorithmic decomposition to use to achieve the highest possible performance for a wide range of problems. The user must also decide on the necessary granularity of the problem. Granularity refers to the amount of time spent computing versus communicating. In course grain problems, large chunks of the code can be worked on independently. Fine grain problems require significant amounts of communication between processors.

2.5 PERFORMANCE MEASURING AND DEGRADATION

Performance is measured in several different ways. The four major standards are: speedup, efficiency, accuracy, and cost-performance ratio. Speedup tells the user how much

faster a particular algorithm runs on n processors when compared to one. It is the ratio of the execution time of the algorithm on a single processor, T_s, to the execution time of the parallel algorithm (solving the same problem) on n processors, T_p:

$$Speedup = \frac{T_s}{T_p}. \qquad (2-1)$$

Speedup has a maximum value of n.

Efficiency tells the user how efficiently the n processors are being used. It is defined as speedup divided by n and expressed as percent (maximum value for efficiency is 100 percent):

$$Efficiency = \frac{Speedup}{n} x\,100\%. \qquad (2-2)$$

Accuracy is defined in the usual sense. A parallel program should provide the same computational accuracy as the corresponding sequential program. Cost-performance refers to the ratio of the processing power to the purchase price for a parallel computer.

The performance of an algorithm on a parallel computer can be degraded by a number of things. Mechanisms that can degrade performance are:

- *Scheduling*: The efficiency with which the available work is distributed among the processors. This is also referred to as load balancing.
- *Communication*: Time spent communicating information between nodes instead of computing.
- *Synchronization*: Idle time spent by a processor waiting for another processor to finish a calculation. Often operations need to take place in a defined order.
- *Duplication*: Work effort that is duplicated in a number of processors. This often occurs when an algorithm cannot be neatly decomposed.

The first three mechanisms cause processors to sit idle; the last mechanism causes processors to waste time doing unnecessary work. These problems cannot be avoided completely, but careful programming can minimize them.

2.6 DRAWBACKS TO PARALLEL PROCESSING

The major drawback to using parallel computers is the amount of work required by the user to fully exploit their potential. Fortunately, with the software that is becoming available in the form of parallel operating environments, such as ParaSoft's Express, this problem is being mitigated. However, it will be a long time before software that will automatically parallelize any but the simplest problems will be available.

Another drawback to using parallel computers is that algorithms developed on one hardware platform may not be easily portable to another hardware platform. Once again, operating environments like Express are helping to mitigate this problem. However, it is still not completely possible to divorce the algorithm from the hardware architecture and still achieve optimum performance.

2.7 THE TRANSPUTER

Of the many parallel computing building blocks (processors) available today, the transputer, originally developed by INMOS (1988), is one of the most cost-effective and flexible. The transputer is actually a computer on a chip. A number of companies have developed inexpensive transputer-based boards that plug into a PC. These boards can each hold up to 10 transputer modules (known as TRAMs) and multiple boards can be connected to provide quite powerful desktop parallel computing platforms.

2.7.1 Transputer Hardware

The transputer uses very large-scale integration (VLSI) technology and is a 32-bit reduced instruction set computer (RISC) design. This makes it a very powerful processor capable of performing a 32-bit floating point multiplication in less than one microsecond. The chip contains a central processing unit (CPU), a floating point processing unit (FPU), 4-KBytes of on-chip static ramdom access memory (RAM), 4 communication links (to host processor or other transputers), and an external memory interface. The communications links, floating point processor and CPU can execute concurrently (allowing simultaneous integer and floating-point computations). Integer only transputers can be purchased, but the floating-point units are more standard for most applications. The standard floating point transputer is the T800. The 20-MHz T800 has a peak performance of 1.5 Mflops.

The transputer is usually packaged with external RAM into a transputer module (TRAM). Many sizes of external RAM are available, but the most common TRAMs have 1-MByte of RAM and are size 1. The size of the TRAM refers to the number of slots it takes up on a motherboard. In other words, a size 1 TRAM requires 1 slot, a size 4 TRAM requires four slots, and so on. The larger size TRAMs contain more RAM. TRAMs are used as nodes in MIMD, distributed memory parallel computers. TRAM based parallel computers require a host computer such as a PC, Vax, or Sun system. A TRAM motherboard fits into any free slot on the host computer. A software reconfigurable crossbar switch is provided to allow reconfiguration of the interconnects between individual processors.

2.7.2 Transputer Software

There are several different ways to program an array of transputers. Each method has tradeoffs in portability, performance, and ease of use. The parallel language, Occam, was developed concurrently with the transputer. Parallel versions of high-level languages such as Fortran, C, Pascal, and Modula 2 can be obtained. The parallel operating environment Express (ParaSoft Corporation, 1990b) is available for transputer systems.

2.7.2.1 Occam. There are a number of advantages to using Occam for programming a transputer system. Since Occam was developed simultaneously with the transputer, it is one of the few languages designed for concurrency. Occam has the performance and efficiency of assembly language. Occam can be used as a harness to link modules written in other industry-standard languages. Its other features include: well-implemented timing capabilities, straightforward control of process scheduling, and a high-level structure.

The main disadvantage to using Occam is that programs written in Occam are not portable to other parallel computers. Existing codes written in other languages need to be completely recoded. Occam works only with the transputer. Other disadvantages are that

Occam does not support many features of other high-level languages (no recursion or dynamic memory allocation), use of it requires learning a new language, and Occam may not be supported in the future. In the tradeoff space (portability versus performance versus ease of use) Occam scores high only in performance.

2.7.2.2 High-Level Languages. The advantage of using a parallel version of a high-level language such as Fortran, C, Pascal, or Modula 2 is that one can use a language with which one is already familiar. All that is required is to learn the parallel extensions. Existing sequential code can be ported over.

There are, however, a number of disadvantages to using a parallel version of a high-level language. Programming an array of transputers requires the user to develop both a host program and a node program. The host program runs on the host processor and handles input/output (I/O) calls and manages the other processors. The node program runs on all the other processors and does the bulk of the computations. The user is required to maintain two programs instead of only one program. This can make program development and maintenance somewhat complex. Performing I/O, making system calls, and timing algorithms can be difficult. Debuggers for parallel versions of high level languages are starting to appear, but they are not widely available. Debugging parallel programs is very difficult. In addition to all this, parallel versions of high-level languages do not offer the performance that one can get from Occam.

2.7.2.3 Express. Parallel operating environments such as ParaSoft's Express allow the user the convenience of using a familiar high-level language while mitigating some of the disadvantages to parallel high-level languages mentioned in the previous section. Express is based on the CUBIX programming model developed at CalTech. Express is available for both Fortran and C. There is no need to develop both a host and node program. Instead, only one program (that runs on all processors) needs to be developed and maintained. Express does not include a compiler so one of the parallel high-level language compilers mentioned above is still needed to compile and link the code. During linking, the Express library is linked into the user's source code. The Express library provides simple function calls to handle I/O and system calls. Express also provides timing capabilities, a source-level debugger, and a performance monitor. The number of processors being used does not have to be hardwired into the code. Instead, this is specified at runtime by a switch in the run command. In addition, Express is available for a wide variety of parallel machines, not just the transputer. Currently Express is available for: the multiheaded IBM 3090 (AIX) and the multiheaded CRAY (UNICOS) mainframes; the Intel iPSC 2 and iPSC 860; the nCUBE 1 and 2; PC, MAC, and SUN hosts for transputers; and the IBM RS/6000, Silicon Graphics, and SUN workstations. All these lead to very portable source code.

The only significant disadvantage to Express is that it possibly degrades system performance to some extent. However, this disadvantage is more than offset by the advantage of having portable code.

3.0 COMPUTATIONAL ELECTROMAGNETICS

Computational electromagnetics (CEM) is that branch of electromagnetics that routinely involves using a computer to obtain results. It is a complementary tool to the classical techniques of experimental observation and mathematical analysis.

The CEM application of interest to the authors of this report is naval ship electromagnetic design. The specific areas of interest include exterior communications, electromagnetic pulse protection, antenna design, and scattering characterization.

3.1 ELECTROMAGNETIC MODEL

The electromagnetic model uses transfer functions derived from Maxwell's equations. The inputs to the transfer functions include a description of the problem as well as the specified excitation. The problem description is defined in terms of both the electrical and the geometrical properties of the structures and the space in which they reside. The specified excitation can be either a voltage source applied to an antenna or a plane wave impeding on the defined structure. The outputs from the transfer functions are the induced currents on the structures and/or the impedance of the antenna. These induced currents can be used to determine both the near and far fields. Near fields are of interest in determining coupling between antennas as well as hazards to personnel, fuel, and ordnance. Far fields are used to determine the performance of a system.

3.2 SHIP EM DESIGN PROCEDURE

This section describes the design procedure used for Navy shipboard exterior radio frequency (RF) communication system design (Li, Logan, & Rockway, 1988). The approach is an iteration process by which candidate RF system designs can be analyzed to determine their relative desirability.

The present procedure for designing a shipboard exterior RF communication system consists of iterations around two loops as shown in figure 3-1. In loop 1, an interactive design model is used to iterate through the various proposed design changes. Given a proposed design, as depicted in an RF system diagram, a designer sets requirements on the space isolation between transmit and receive antennas. By comparing the required antenna isolation with the achieved antenna isolation, the design determines whether additional antenna isolation is needed between transmit and receive antennas to obtain a compatible system. The achieved antenna isolation value is obtained from the topside antenna study.

The output at the intermediate level shown in figure 3-1 is an intermediate design along with the power and frequency spectrum constraints required for compatible use of the platform transmit and receive subsystems. After the designer decides that significant additional improvements either are not needed, or are not likely to be found, a performance evaluation study followed by a link study is performed. The performance measures used are the time availability of a given quality of service for a given signal and the resulting compatibility grade. In a case where the design is not acceptable, further modifications to the design can be made using loop 2 and/or 1 of figure 3-1.

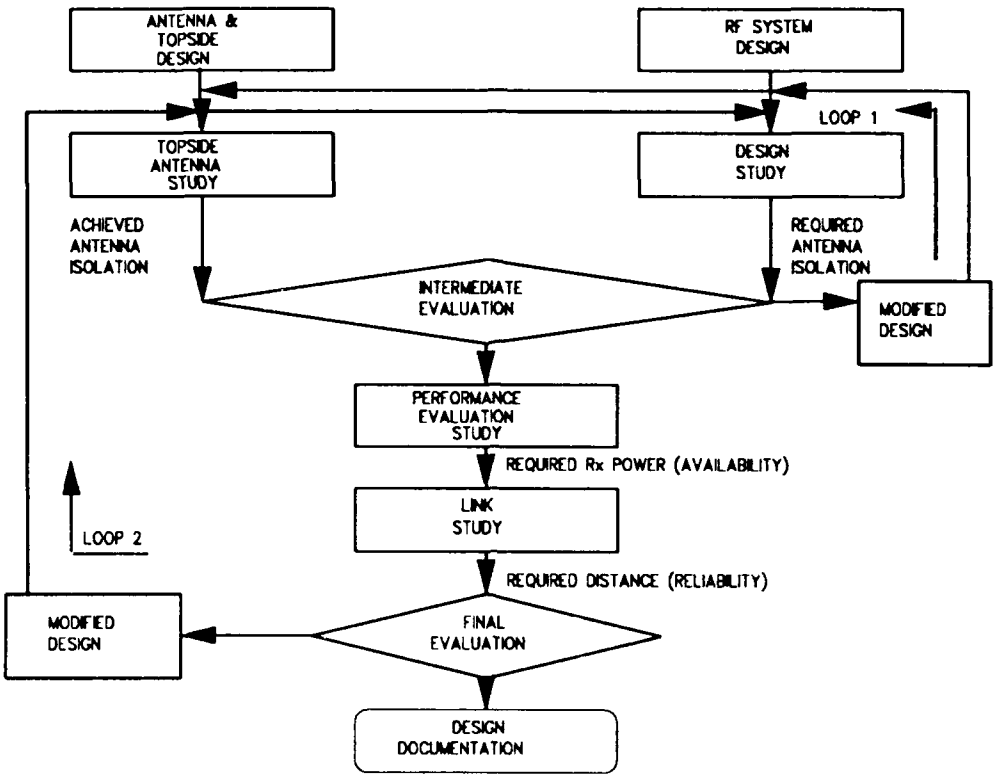


Figure 3-1. Design procedure for a shipboard exterior RF communication system.

3.3 TOPSIDE ANTENNA STUDY

The objective of the topside antenna study is to determine the technical parameters of the topside antenna element. The technical parameters of an antenna element are completely specified by the impedance, near fields, radiation pattern, and coupling to other antenna elements. These technical parameters are necessary to determine the performance of the total shipboard exterior communication system.

At the present time, there exist two techniques for performing a topside antenna study: scale-brass modeling and numerical modeling using the numerical electromagnetic code (NEC). Because of the complexity of the systems involved, the application of analytical techniques has very limited utility.

3.3.1 Brass Modeling

Scale-brass ship models are used on a model range to do experimental observations using radiation which is also scaled in frequency. There are a number of drawbacks to using scale models. Scale models are time-consuming to build and difficult to modify. It is very difficult to measure near fields for scale models. Worst of all, scale models are limited to

perfect conductors since nonperfect conductors have frequency-dependent electric and magnetic properties. The Navy is planning on using composite materials for the next generation of Navy ships.

3.3.2 Numerical Electromagnetic Code

The numerical electromagnetic code (NEC) was developed for antenna modeling. NEC includes the NEC-method of moments (NEC-MoM), NEC-basic scattering, and NEC-reflector antenna codes (Li, et al., 1988). All these codes have been validated and extensively documented. NEC-MoM was developed at the Lawrence Livermore National Laboratory (LLNL). The NEC-basic scattering code and the NEC-reflector antenna code were developed at Ohio State University.

At the lower frequencies of interest (generally below 30 MHz), NEC-MoM yields estimates of the technical parameters of antennas mounted on ships. At the intermediate and upper frequency ranges, the NEC-basic scattering and NEC-reflector antenna codes, based on the geometric theory of diffraction (GTD) provide useful information on shipboard antenna elements.

NEC-MoM is the most advanced computer code available for the analysis of thin wire antennas. It is a highly user-oriented computer code offering comprehensive capability for the analysis of the interaction of electromagnetic waves with conducting structures. The program is based on the numerical solution of integral equations for the currents induced on the structure by an exciting field.

NEC-MoM combines integral equations for smooth surfaces with one for wires to provide convenient and accurate modeling of a wide range of applications. The NEC-MoM model may include nonradiating networks and transmission lines, perfect and imperfect conductors, lumped element loading, and ground planes. The ground plane may be perfectly or imperfectly conducting. Excitation may be via an applied voltage source or incident plane wave. The output may include currents and charges, near and far zone electric or magnetic fields, and impedance or admittance. Many other commonly used parameters such as gain and directivity, power budget, and antenna-to-antenna coupling are also available.

The NEC-basic scattering code (BSC) is a user-oriented computer code for the analysis of the Fresnel region fields and the far-field patterns of antennas in the presence of perfectly conducting metal structures at intermediate and upper frequency ranges. Complicated structures can be simulated by arbitrarily oriented flat plates, an infinite ground plane, and a finite elliptic cylinder. The analysis is based on uniform asymptotic techniques formulated in terms of the GTD. The GTD approach is useful for a general high-frequency study of antennas in a complex environment involving structures that are large with respect to the wavelength in that only the most basic structural features of an otherwise very complicated structure need to be modeled.

Modeling electrically small structures and antennas can be better accomplished using an integral-equation solution such as NEC-MoM. The basic scattering code has been interfaced with the MoM code so that the capabilities of both methods are used optimally.

The NEC-reflector antenna code was designed to compute either near-field curves or far-field patterns of typical Navy reflector antennas. One important feature of the code is the capability to input a practically arbitrary volumetric feed pattern. The theoretical approach for computing the fields of the general reflector is based on a combination of the GTD and aperture integration techniques.

3.4 FUTURE OF COMPUTATIONAL ELECTROMAGNETICS

The drawback to using computational electromagnetics codes such as NEC is the amount of effort required to develop the input, perform calculations, and interpret the output. Preparing the input for NEC and evaluating the output can be an overwhelming task. Performing calculations can be exceedingly time consuming for all but the most simple structures. This drawback is being addressed in three areas. Design systems such as the numerical electromagnetic engineering design system (NEEDS) are being developed to decrease the amount of effort required to define input and interpret output. Efficient computational methods are being used to decrease the effort required to perform calculations. Finally, accurate and more useful algorithms (such as those being developed at the University of Houston) are being used to improve calculations.

3.4.1 NEEDS

From a user's point of view, the most important aspects of a MoM computer code are the effort to prepare input data and verify its correctness, the complexity and size of problems that a given code can solve, the computer resources it requires, and the accuracy that it provides. Many of these user requirements were being addressed by the NOSC development of the numerical electromagnetic engineering design system (NEEDS) under Office of Naval Technology (ONT) sponsorship. A major portion of the NEEDS capability is a workstation to assist the user in making the modeling process less tedious and more error-free. The modeling process essentially has three parts: a. Problem definition consists of defining the geometry of the problem, specifying the electromagnetic parameters such as frequency and voltage sources, and deciding on controls such as what outputs to produce, which algorithms to use and whether to perform cost analysis. b. Processing is performed using the NEC algorithms. c. Solution description consists of thought-enhancing displays and reports as well as audit trails and data management and analysis. In the future, NEEDS will use expert systems and graphics techniques to automate the modeling process and guide the engineer from the preparation of the problem definition through solution description. NEEDS will ensure compliance to the design procedure, fast and accurate problem definition, and thought-enhancing output products.

3.4.2 Efficient Computational Methods

The overall effort from problem formulation through discretization and solution of the resulting complex matrices is compute-bound and expensive. The computer resources required for a given problem are critical to a user. With the advent of improved MoM algorithms such as those described below and the development of NEEDS, it is the computing machine that is becoming the limiting factor in the use of numerical modeling for synthesizing antenna designs. Developments have concentrated on the implementation of solvers on

conventional computing machines. This report documents research investigating the use of parallel processing in MoM. The emphasis was on determining if the computing machine limitation to computational electromagnetics could be mitigated.

3.4.3 Advanced Algorithms

From a code developer's point of view, the most important aspects of a computer code are the formulation, numerical implementation, and the accuracy that is provided. Recently, advanced MoM algorithms have been developed by Professor Donald Wilton at the University of Houston (Hwu and Wilton, 1988). Under ONT sponsorship these algorithms are presently being extended by Professor Wilton to the modeling of composite surfaces. The Wilton (1988) algorithms invoke the MoM to solve a coupled electric field integral equation for the currents induced on an arbitrary configuration of perfectly conducting surfaces and wires. This formulation involves defining the unknown current at points on the surface using a triangular function expansion and then sampling the formulation using weight functions as defined by the Galerkin technique.

The Wilton formulation described above is known as the JUNCTION formulation because of its ability to handle junctions between bodies and wires. JUNCTION provides more uniform surface current representation than previous versions of MoM, and will be able to evaluate magnetic and dielectric surfaces.

3.5 THEORY OF JUNCTION

The JUNCTION computer code invokes the MoM to solve a coupled electric field integral equation (EFIE) for the currents induced on an arbitrary configuration of perfectly conducting bodies and wires. NEC models conducting bodies either as a wire mesh or as a surface of magnetic field integral equation (MFIE) patches.

Principal advantages of the wire grid approach are that the geometry is easily specified for computer input and only one-dimensional integrals need to be evaluated. The wire grid modeling approach, however, often proves unsatisfactory where near-field quantities such as surface currents or input impedance are desired. One obvious difficulty is in interpreting computed wire currents as equivalent surface currents. Also, the storage of energy in the neighborhood of a wire mesh is not completely equivalent to that of a continuous surface. As a result, computed resonant frequencies and reactive components of computed impedances are often shifted from their correct values. Modeling using the MFIE patch is limited to surfaces that are closed and are smooth and, therefore, lacks application to many "real world" problems.

There are three principal advantages to the JUNCTION formulation. First, the EFIE formulation for the surfaces of bodies, in contrast to the magnetic field integral equation (MFIE), applies to open bodies. Second, JUNCTION allows voltage and load conditions to be easily specified at terminals defined on the structure. Third, the triangular patches of JUNCTION are the simplest planar surfaces that can be used to model arbitrary surfaces and boundaries, and triangular patches permit patch densities to be varied locally so as to model a rapidly varying current distribution.

3.5.1 Formulation of JUNCTION

Figure 3-2 shows a sample geometrical structure of a body with perfectly conducting wires near and connected to a perfectly conducting body. For this general structure, S is used to denote the surface of the perfectly conducting bodies and wires. S_B denotes the surfaces of the conducting bodies, and S_W denotes the surfaces of the conducting wires. It will be assumed that this general structure is immersed in an incident electromagnetic field, $\overline{E}^i$. A pair of coupled integral equations for this general structure may be derived by requiring the tangential component of the electric field to vanish on each surface. Thus,

$$\overline{E}^i_{\tan} = (j\omega\overline{A} + \nabla\Phi)_{\tan}. \tag{3-1}$$

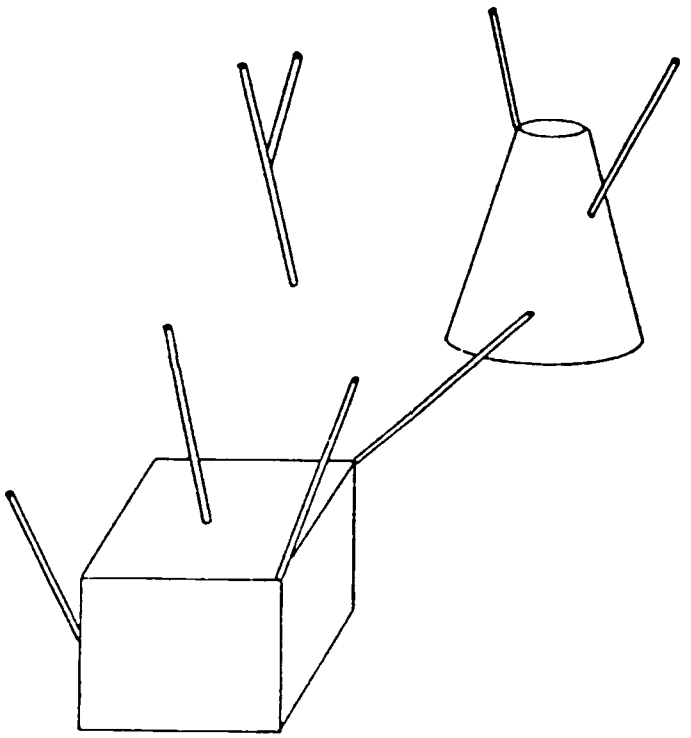


Figure 3-2. Sample geometrical structure of a collection of conducting bodies and wires.

The vector potential, $\overline{A}$, is given by

$$\overline{A} = \frac{\mu}{4\pi}\left[\int_{S_B} G(R)\,\bar{J}\,dS' + \int_{S_W} \frac{I\hat{s}'G(R)}{2\pi a(s')}\,dS'\right] \tag{3-2}$$

and the scalar potential Φ by

$$\Phi = -\frac{1}{j4\pi\omega\varepsilon}\left[\int_{S_B} \nabla' \cdot \bar{J} G(R) dS' + \int_{S_W} \frac{1}{2\pi a(s')} \frac{dI}{ds'} G(R) dS'\right] \qquad (3-3)$$

where

$$G(R) = \frac{e^{-jkR}}{R}.$$

$R = |\bar{r} - \bar{r}'|$ is the distance between an arbitrarily located observation point at $\bar{r}$ and a source point at $\bar{r}'$ on a perfectly conducting surface, S. s' is the arc length along the wire axis in the direction denoted by the unit vector $\hat{s}$. The radius of the wire is denoted by a and is a function of the arc length s'. $\bar{J}$ is the surface current on the body, and I is the current along the wire. In equations 3-1 through 3-3, $k = \frac{2\pi}{\lambda}$, where λ is the wavelength. ω is the radian frequency. μ and ε are permittivity and permeability, respectively, of the surrounding medium.

3.5.2 Numerical Procedure

The MoM is a numerical procedure for solving the electric field integral equation 3-1 (Harrington, 1968). Basis functions are chosen to represent the unknown currents. Testing functions are chosen to enforce the integral equation on the surface of the conducting structure. With the choice of basis and testing functions a matrix approximating the integral equation is derived.

3.5.2.1 Basis Function. Three different sets of basis functions are used to represent the current on a general structure. One basis function represents the currents induced on bodies. Another basis function represents the currents induced on wires. A third basis function represents the current in the neighborhood of wire-to-surface junctions. The basis functions must be linearly independent and capable of approximating the actual surface current. As shown in figure 3-3, a triangular patch model is used for the bodies. The basis function used in JUNCTION for representing currents induced on the bodies (S_B) is given by

$$\bar{\Lambda}_n^B(\bar{r}) = \frac{\bar{\rho}^{n\pm}}{h_n^{B\pm}} \qquad (3-4)$$

where $S_n^{B\pm}$ is the $\pm$ reference triangle attached to the nth nonboundary edge of a body. The current at a nonconnected edge is zero. The distances from vertex of the triangle opposite the nth edge, the free vertex, to the nth edge of S_B are $h_n^{B\pm}$, and $\bar{\rho}^{n\pm}$ is the vector from the free vertex of $S_n^{B\pm}$ to the vector $\bar{r}$ intersecting at the surface S_B. The surface divergence of $\bar{\Lambda}_n^B(\bar{r})$ is

$$\nabla \cdot \bar{\Lambda}_n^B(\bar{r}) = \pm\frac{2}{h_n^{B\pm}}. \qquad (3-5)$$

As shown in figure 3-4, a tubular segment model is assumed for wires. The basis function used in JUNCTION for representing currents induced on the wires (S_W) is given by

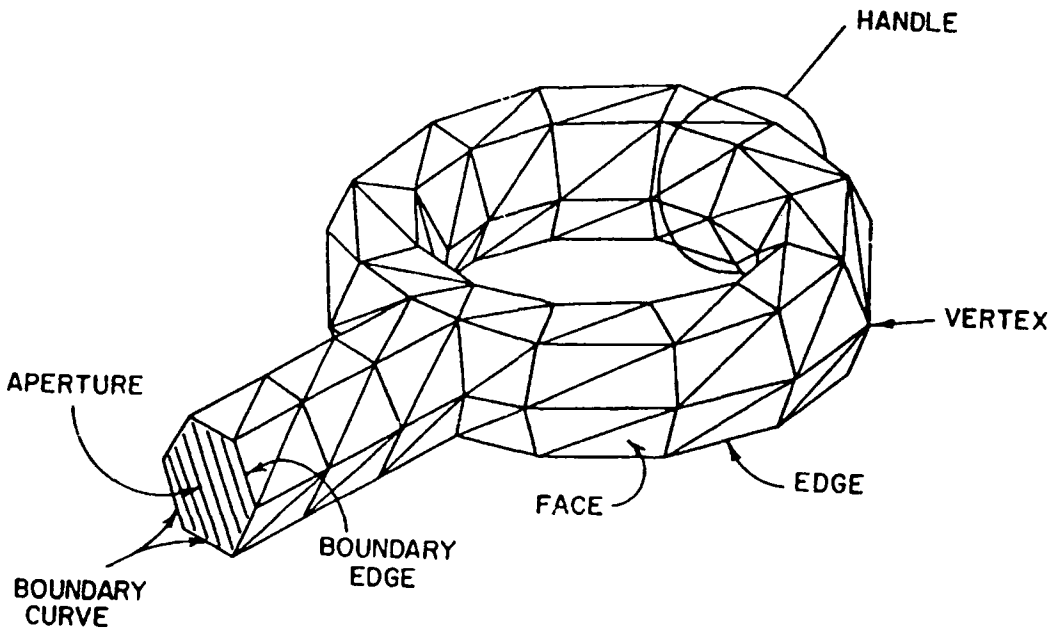


Figure 3-3. Arbitrary surface modeled by triangular patches.

$$\bar{\Lambda}_n^W(\bar{r}) = \frac{\bar{\rho}^{\pm}}{h_n^{W\pm}} \tag{3-6}$$

where $S_n^{W\pm}$ is the $\pm$ reference segment attached to the nth nonboundary node of a wire. The length of $S_n^{W\pm}$ relative to the nth node of S_W is $h_n^{W\pm}$, and $\bar{\rho}^{\pm}$ is the vector from the free node of $S_n^{W\pm}$ to the vector $\bar{r}$. The surface divergence of $\bar{\Lambda}_n^W(\bar{r})$ is

$$\nabla \cdot \bar{\Lambda}_n^W(\bar{r}) = \pm \frac{1}{h_n^{W\pm}}. \tag{3-7}$$

Finally, a basis function is associated with wire junctions on the body. A wire-to-surface junction is assumed to exist only at a triangle vertex. Referring to figure 3-5, the vector basis function associated with the nth junction is

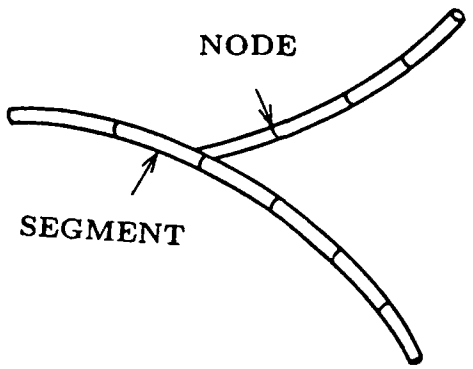


Figure 3-4. Arbitrary wire modeled by tubular segments.

$$\overline{\Lambda}_n^J(\bar{r}) = K_{nl}\left[1 - \frac{(h_{nl}^{J+})^2}{(\rho'^{+} \cdot \hat{h}_{nl}^{J+})^2}\right]\overline{\Lambda}_{nl}^B(\bar{r}) \qquad (3-8)$$

and

$$\overline{\Lambda}_n^J(\bar{r}) = \overline{\Lambda}_n^W(\bar{r}). \qquad (3-9)$$

The double index nl refers to the lth triangular patch on the body surface at the nth junction. $\overline{\Lambda}_{nl}^B(\bar{r})$ and $\overline{h}_{nl}^{J+}$ are the previously defined body basis functions and the vector distance respectively. These vectors are associated with the edge opposite the junction vertex S_{nl}^{J+}. The total flux from the junction triangles into the wire is normalized to unity if

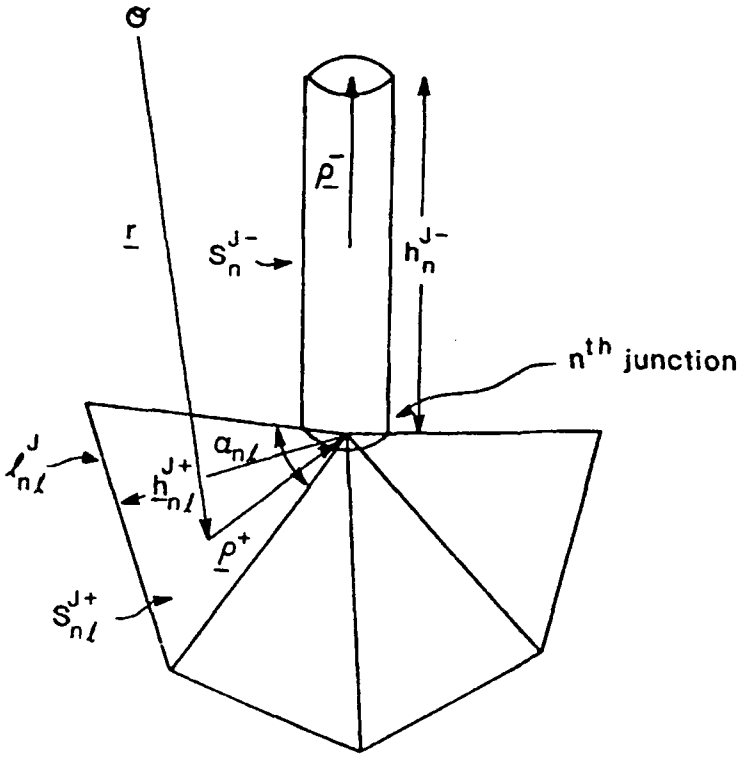


Figure 3-5. Geometrical parameters associated with the *n*th wire-to-surface junction.

$$K_{nl} = \frac{\alpha_{nl}}{l_{nl} \sum_{i=1}^{N_{jn}} \alpha_{nl}} = \frac{\alpha_{nl}}{l_{nl} \alpha_n^t} \tag{3-10}$$

where α_{nl} is the angle between the two edges of S_{nl}^{J+} common to the nth junction vertex. l_{nl} is the length of the edge opposite S_{nl}^{J+}, and α_n^t is the sum of the nth junction vertex angles. N_{jn} is the number of patches attached to the nth junction. The surface divergence $\overline{\Lambda}_n^J(\bar{r})$ is derived from

$$\nabla_s \cdot \overline{\Lambda}_n^J(\bar{r}) = 2 \frac{K_{nl}}{h_{nl}^{J+}} \tag{3-11}$$

where $\bar{r}$ intersects S_{nl}^{J+} and

$$\nabla_s \cdot \overline{\Lambda}_n^J(\bar{r}) = -\frac{1}{h_n^{J-}} \tag{3-12}$$

where $\bar{r}$ intersects S_n^{J-}.

The current on the surfaces of the bodies S_B may now be represented as

$$\overline{J}(\bar{r}) \approx \sum_{n=1}^{N_B} I_n^B \overline{\Lambda}_n^B(\bar{r}) + \sum_{n=1}^{N_J} I_n^J \overline{\Lambda}_n^J(\bar{r}) \tag{3-13}$$

and the total axial current on the wires S_W may be represented as

$$I(\bar{r})\hat{l} \approx \sum_{n=1}^{N_W} I_n^W \overline{\Lambda}_n^W(\bar{r}) + \sum_{n=1}^{N_J} I_n^J \overline{\Lambda}_n^J(\bar{r}) \tag{3-14}$$

where N_B, N_W, and N_J are the number of unknown currents on the bodies, wires, and junctions respectively. Since the surface divergences are proportional to the charge density, the charge is constant on all body, wire, and junction subdomains.

3.5.2.2 Testing. The next step in applying the MoM is to select a suitable testing procedure. Referring to figure 3-6, the integral equation on S_B is enforced by integrating the vector component of equation 3-1 parallel to the path from the centroid of S_m^{B+} to the middle of the edge l_m^B and then to the centroid of S_m^{B-}. Similarly, the integral equation on S_W is enforced by integrating the vector component of equation 3-1 parallel to the path from the centroid of S_m^{W+} to node m and next to the centroid of S_m^{W-}. At a junction, the tangential electric field along a path from the centroid of each junction triangle is integrated to the junction and then along the wire axis to the center of the attached wire segment. The resulting equations are then combined into a single equation for the junction by weighting each with the associated triangle vertex angle, and summing the results for each junction patch. For all path integrals, $\overline{E}^i$ and $\overline{A}$ are approximated along each portion of the path by their respective values at the centroids. The integral on $\nabla\Phi$ reduces to a difference of scalar potentials at the path end points. This eliminates the requirement implied in equation 3-1 that Φ be differentiable. Thus, for bodies equation 3-1 becomes

$$j\omega\left[\overline{A}(\bar{r}_m^{B+}) \cdot \bar{l}_m^{B+} + \overline{A}(\bar{r}_m^{B-}) \cdot \bar{l}_m^{B-}\right] + \left[\Phi(\bar{r}_m^{B-}) - (\Phi(\bar{r}_m^{B+})\right]$$

$$= \overline{E}^i(\bar{r}_m^{B+}) \cdot \bar{l}_m^{B+} + \overline{E}^i(\bar{r}_m^{B-}) \cdot \bar{l}_m^{B-} \tag{3-15}$$

for $m = 1, 2, \ldots, N_B$.

For wires, equation 3-1 becomes

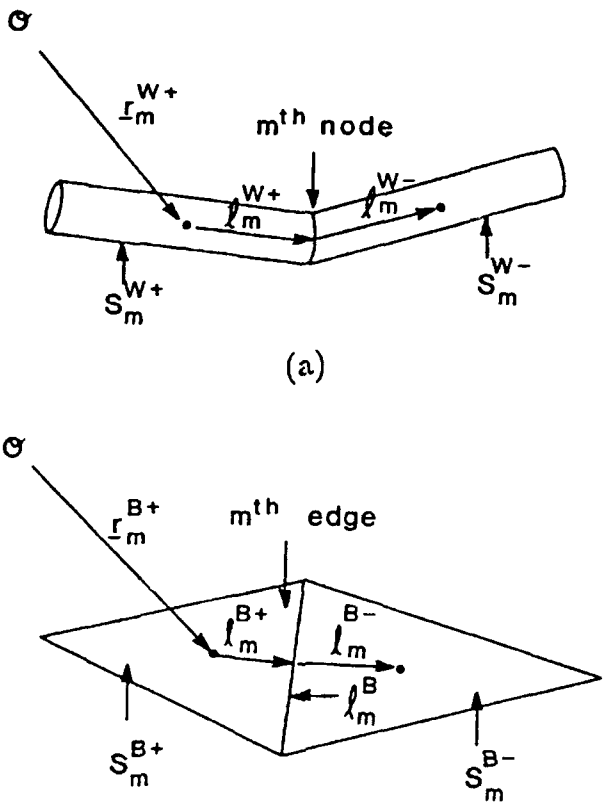


Figure 3-6. (a) Testing path associated with the mth node of wire.
(b) Testing path associated with the mth edge of body.

$$j\omega\left[\overline{A}(\bar{r}_m^{W+})\cdot\bar{l}_m^{W+}+\overline{A}(\bar{r}_m^{W-})\cdot\bar{l}_m^{W-}\right]+[\Phi(\bar{r}_m^{W-})-(\Phi(\bar{r}_m^{W+})]$$

$$=\overline{E}^i(\bar{r}_m^{W+})\cdot\bar{l}_m^{W+}+\overline{E}^i(\bar{r}_m^{W-})\cdot\bar{l}_m^{W-} \qquad (3-16)$$

for $m = 1,2,\ldots,N_W$.

For junctions, equation 3-1 becomes

$$\frac{1}{\alpha_m^t}\sum_{l=1}^{N_{Jm}}\alpha_{ml}[j\omega\overline{A}(\bar{r}_{ml}^{J+})\cdot l_{ml}^{J+}-\Phi(\bar{r}_m^{J+})]+j\omega\overline{A}(\bar{r}_m^{J-})\cdot\bar{l}_m^{J-}+\Phi(\bar{r}_m^{J-})$$

$$=\frac{1}{\alpha_m^t}\sum_{l=1}^{N_{Jm}}\alpha_{ml}\overline{E}^i(\bar{r}_m^{J+})\cdot\bar{l}_m^{J+}+\overline{E}^i(\bar{r}_m^{J-})\cdot\bar{l}_m^{J-} \qquad (3-17)$$

for $m = 1,2,\ldots,N_J$.

For plane wave scattering problems

$$\overline{E}^i(\overline{r}) = (E_\theta^i \hat{\theta}^i + E_\phi^i \hat{\phi}^i) e^{j\overline{k}\cdot\overline{r}} \tag{3-18}$$

where the propagation vector $\overline{k}$ is

$$\overline{k} = k(\sin\theta^i \cos\phi^i \hat{x} + \sin\theta^i \sin\phi^i \hat{y} + \cos\theta^i \hat{z}) \tag{3-19}$$

and $(\hat{\theta}^i, \hat{\phi}^i)$ define the angle of arrival in the usual spherical coordinate convention and can be expressed as

$$\hat{\theta}^i = \cos\theta^i \cos\phi^i \hat{x} + \cos\theta^i \sin\phi^i \hat{y} - \sin\theta^i \hat{z} \tag{3-20}$$

$$\hat{\phi}^i = -\sin\phi^i \hat{x} + \cos\phi^i \hat{y}. \tag{3-21}$$

For antenna radiation problems $E_\theta^i = E_\phi^i = 0$. In this case, the wire segments attached to a node or junction m may be thought of as separated by a gap across which the voltage is specified.

3.5.2.3 Evaluation of Matrix Elements. Substituting the current representations of equations 3-13 through 3-14 into equations 3-15 through 3-17, a system of N linear equations results, where N = NB + NW + NJ.

$$\begin{bmatrix} [Z^{BB}] & [Z^{BW}] & [Z^{BJ}] \\ [Z^{WB}] & [Z^{WW}] & [Z^{WJ}] \\ [Z^{JB}] & [Z^{JW}] & [Z^{JJ}] \end{bmatrix} \begin{bmatrix} [I^B] \\ [I^W] \\ [I^J] \end{bmatrix} = \begin{bmatrix} [E^B] \\ [E^W] \\ [E^J] \end{bmatrix} \tag{3-22}$$

where Z is the impedance matrix, I is the current vector, and E is the excitation vector. The superscripts are B for body elements, W for wire elements, and J for junctions between wires and bodies. The elements of the submatrices are given by

$$Z_{mn}^{\gamma\beta} = jw[A_{mn}^{\gamma\beta+} \cdot l_m^{\gamma+} + A_{mn}^{\gamma\beta-} \cdot l_m^{\gamma-}] + [\Phi_{mn}^{\gamma\beta-} - \Phi_{mn}^{\gamma\beta+}], \qquad \gamma \neq J,$$

$$Z_{mn}^{J\beta} = \frac{1}{\alpha_m^l} \sum_{l=1}^{N_{jm}} \alpha_{ml} (j\omega A_{mln}^{J\beta+} \cdot l_{ml}^{J+} - \Phi_{mln}^{J\beta+}) + j\omega A_{mn}^{J\beta-} \cdot l_m^{J-} + \Phi_{mn}^{J\beta-},$$

$$A_{mn}^{\gamma\beta\pm} = A_n^\beta(r_m^{\gamma\pm}), \qquad \Phi_{mn}^{\gamma\beta\pm} = \Phi_n^\beta(r_m^{\gamma\pm}),$$

$$A_{mln}^{J\beta+} = A_n^\beta(r_{ml}^{J+}), \qquad \Phi_{mln}^{J\beta+} = \Phi_n^\beta(r_{ml}^{J+}),$$

$$A_n^\beta(r) = \frac{\mu}{4\pi} \int_{S_\beta} \Lambda_n^\beta(r') \frac{e^{-jkR}}{R} dS'$$

$$\Phi_n^\beta(r) = -\frac{1}{j4\pi\omega\varepsilon}\int_{S_\beta} \nabla_s \cdot \Lambda_n^\beta(r')\frac{e^{-jkR}}{R}dS'.$$

Solution of the linear system of equations yiel. the set of unknown coefficients used in the representation of the surface, wire, and junction currents, equations 3-13 and 3-14. Once these currents are known, the scattered field or any other electromagnetic quantity of interest may be determined using standard matrix solution routines. JUNCTION uses the LINPACK subroutines for solving the complex matrix of equation 3-22 using Gaussian elimination.

4.0 APPROACH

There were two objectives to this IED investigation. The principal objective was to demonstrate that parallel processing could improve the utility and efficiency of computational techniques used in ship electromagnetic design. A secondary objective was to permit the authors to become familiar with parallel processing techniques and computing platforms. In light of these objectives the following approach was developed.

4.1 INITIAL PLATFORM

The initial hardware platform chosen was an array of four transputers on a motherboard that could be installed inside a PC. This decision was based on the low cost, ease of use, and flexibility of a transputer-based system. Transputer systems provide an inexpensive entry into the world of parallel processing. For this entry-level application, a decision was made to purchase only four floating point TRAMs each with one Megabyte of external RAM. These size 1 TRAMs are an industry standard and provide sufficient memory for most applications at a reasonable cost and in a compact package.

The four TRAMs plus a motherboard to fit into a PC were purchased from Transtech Parallel Systems Corporation (120 Langmuir Laboratory, 95 Brown Road, Cornell Business and Technology Park, Ithaca, NY 14850-9430. (607) 257-6502, FAX (607) 257-3980). The TRAMs were Transtech model TTM3-85-F ($875 ea). The motherboard was Transtech model TMB08 ($732). This motherboard features ten TRAM slots, plugs directly into a free slot on an IBM PC AT or XT and compatibles, includes an IMSC004 link crossbar switch for configurable reconnectivity and comes with network configuration software. A photo of the motherboard with the four TRAMs installed in it is shown in figure 4-1. Additional TRAMs could be added by simply plugging them in.

4.2 OPERATING ENVIRONMENT

Express by ParaSoft was chosen for the operating environment for the reasons discussed in section 2.6.2 of this report. Portability was the key feature since the ultimate goal of this project was to run the code on a high performance computer. The language chosen was Fortran since this is the principal language of existing computational electromagnetics codes. For obvious reasons, it was desirable to make as few changes to the existing code as possible.

The PC version of the ParaSoft parallel computing system for Fortran users (Part No. PS-EXP-F) including parallel environment and parallel monitor were purchased for $1500. (ParaSoft Corporation, 27415 Trabuco Circle, Mission Viejo, CA 92692. (714)-380-9739) A compiler is not included and must be purchased separately. The 3L parallel Fortran compiler was purchased from ParaSoft for $800.

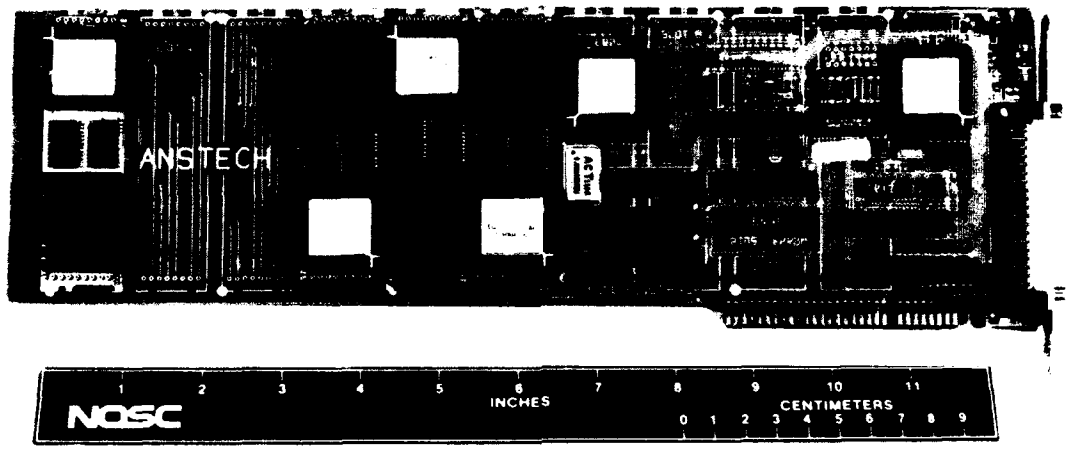


Figure 4-1. Photo of Transtech model TMB08 motherboard with four T800 TRAMs.

4.3 ALGORITHM

The computational electromagnetics algorithm chosen to be parallelized was the JUNCTION code written by the University of Houston. This algorithm is discussed in some detail in section 3.5. It is the most recent MoM algorithm available. JUNCTION is written in Fortran.

Before deciding to parallelize any code it must be determined whether the nature of the code lends itself to being parallelized. This is discussed in section 2.4. JUNCTION appeared to have all the properties indicating it was suitable for running on a parallel computer. Initially, it was unclear whether domain decomposition or algorithmic decomposition (or some combination of the two) would be the preferred method of parallelizing JUNCTION.

4.4 FINAL PLATFORM

After parallelizing the JUNCTION code on the transputer array, the resulting parallel code would be ported over to a parallel high performance computer. This would be necessary to fulfill the main objective of the IED project, namely, to demonstrate improvements to

computational electromagnetics utility. The array of transputers is limited in memory (1 Megabyte per processor) such that only arrays of dimensions less than 500x500 can be solved. A parallel high performance computer would not have this limitation.

NOSC Code 70 is purchasing ParaSoft Express for their iWarp computer. This makes the iWarp well-suited for running the parallelized JUNCTION code. Unfortunately, as of June 1991, the iWarp has still not arrived, and Express is not ready for it. Worst of all, the processors initially being installed do not support Fortran. Processors which will support Fortran will be installed sometime in the next fiscal year, so if the iWarp is to be used for the high- performance computing platform it will have to wait until next year. As of June 1991, an suitable alternative final platform has not been identified.

5.0 IMPLEMENTATION OF SEQUENTIAL PROGRAM

The first step in parallelizing an existing program is to verify that the sequential version of the program works. This is done by running the program on a serial computer, in this case the PC and a microVax. Once this has been successfully done, the next step is to get the program to run on a single processor on the parallel computer. This is necessary to eliminate any problems caused by compiler differences between the serial compiler and the parallel compiler and also to check out the hardware. Only after this has been completed can the parallelization of the program be considered.

5.1 JUN7

In October 1990, the authors of this report visited Professor Donald Wilton at the University of Houston. He gave us the latest version of the JUNCTION code, JUN7.FOR. He also gave us three example problems. The size of JUN7.FOR is 366938 bytes.

JUN7 was compiled and run on the microVax on all three examples. No problems were encountered. Several problems were encountered when trying to run JUN7 on the PC using Microsoft Fortran Version 5.0. The source file needed to be broken up into several smaller files. The values in the parameter statements for the array dimensions needed to be decreased. "SAVE" statements are not needed with Microsoft Fortran and resulted in compile errors. These statements needed to be commented out. Large matrices (greater than 64k) needed to be declared [HUGE] in each of the subroutines in which they were passed as an argument. If this was not done, the matrix writes over on itself, resulting in incorrect output. There are two matrices ETH and EPH, which are both [100x100] complex matrices and can be eliminated. Decreasing the size of these arrays makes the executable code much smaller. The stack size needed to be increased by using the linker switch /F C00. After these changes were made the program could be run on the PC using Microsoft Fortran Version 5.0 on all three examples. The largest problem that could be run on the PC within the 640k limit has 225 unknowns.

5.2 JUNCTION

JUN7 was broken into four separate programs: DATGN, GEOMETRY, CURRENT, and FIELD. DATGN creates input data set files. GEOMETRY calculates the geometrical parameters. CURRENT computes the currents and impedance. FIELD computes charges, near fields, far fields, radar cross sections, and power gain.

The source files, input files, and output files for each program are shown in table 5-1.

The computationally intense programs are CURRENT and FIELD. A decision was made to initially only put CURRENT on the transputers. Later on, if time allowed, FIELD would also be parallelized.

Table 5-1. Breakdown of JUNCTION Code.

PROGRAM NAME	SOURCE FILES	INPUT FILES	OUTPUT FILES
DATGN	DATGN.FOR		FOROO1 (FREQUENCY DATA SET) FOR002 (BODY DATA SET) FOR008 (WIRE DATA SET)
GEOMETRY	GEOMETRY.FOR STRUCT.FOR SUPPORT.FOR	FOR001 FOR002 FOR008	FOR003 (BODY DESCRIPTION) FOR009 (WIRE DESCRIPTION) FOR015 (COMPLETE GEOMETRY DESCRIPTION)
CURRENT	CURRENT.FOR LINPK.FOR ZPK.FOR POTWR.FOR POTBD.FOR SUPPORT.FOR	FOR001 FOR015	FOR004 (COMPUTED CURRENTS)
FIELD	FIELD.FOR CHARGE.FOR PATTERN.FOR NEAR.FOR POTWR.FOR POTBD.FOR SUPPORT.FOR	FOR001 FOR004 FOR015	FOR010 (COMPUTED FAR FIELDS) FOR011 (COMPUTED CHARGE DENSITY) FOR016 (COMPUTED NEAR FIELDS)

5.2.1 CURRENT

From a computational standpoint, CURRENT consists of filling the impedance matrix and the excitation vector shown in equation 3-22, and then solving for the current vector. The current vector gives the currents on the bodies, wires and junctions. The methods used are standard matrix manipulation techniques. Once the matrix is filled, Gaussian elimination is performed using LU decomposition with partial pivoting to factor the matrix. Finally the current vector is solved for by using backward and forward substitution. The computational time for filling the matrix elements is proportional to N squared where N is the dimension of the matrix. The computational time for factoring the matrix is proportional to N cubed.

5.3 TRANSPUTER AND EXPRESS INSTALLATION

The installation of the transputer motherboard with four TRAMs into the PC was straightforward. The board passed all the diagnostic tests provided with it.

Installing ParaSoft Express and the 3L parallel Fortran compiler was also straightforward. Express required about four megabytes of space on the hard disk. The parallel compiler required about 1.5 megabytes. The transputer board passed the Express diagnostics. Express can be operated either under DOS or under Microsoft Windows. DOS usage was sufficient for this project.

A demo program that comes with the 3L compiler was run. This program, which can be run on up to four transputers, generates the Mandelbrot set and displays it. For comparison purposes the program was also run on a single transputer. The execution time on the single transputer is 32 seconds. On the four transputers the execution time is 10 seconds. This is a speedup of 3.2, an efficiency of 80 percent.

5.3.1 Compiling and Executing

Compiling a program and executing it on the transputer array under Express is very straightforward. The 3L Fortran parallel compiler is invoked with a command such as

```
tfc -g -o noddy noddy.f -lcubix
```

where "noddy.f" is the source file. Only one command is required to compile and link the program, creating the executable "noddy" and linking the *Cubix* libraries.

The Express kernel is loaded with the single command

```
exinit
```

If all goes well a display similar to that of figure 5-1 should appear. The actual location in memory in which Express is loaded is determined by the configuration file.

The execution of the program on the array of transputers is achieved simply by executing the *cubix* command. In the simplest case only two pieces of information are needed: the name of the program to execute and the number of nodes on which to execute it. The command

```
cubix -n4 noddy
```

loads the program "noddy" into four transputers and executes it there. The *cubix* command has many options but most of these are seldom used.

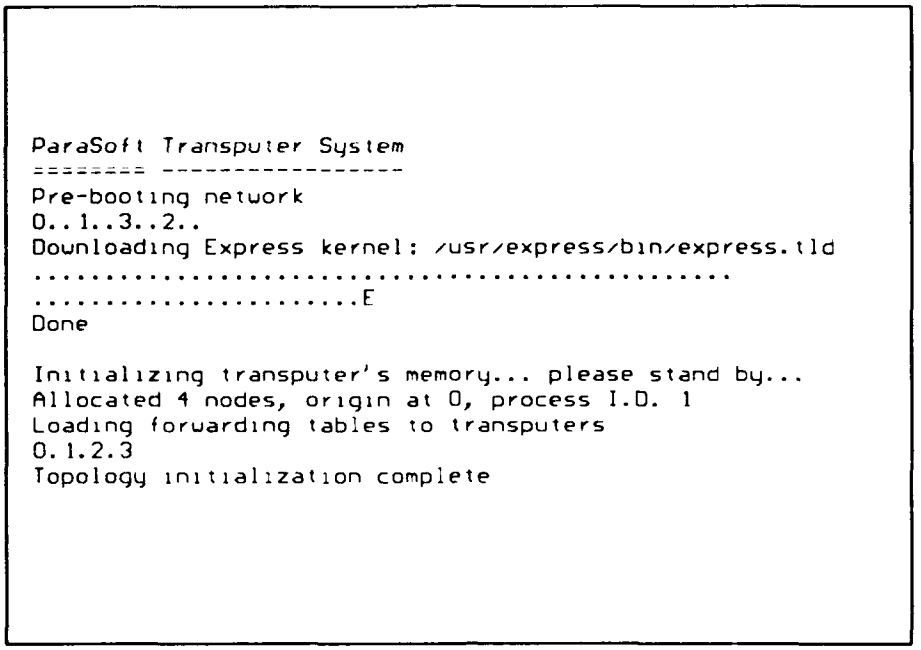


Figure 5-1. Screen display after loading Express kernel.

5.3.2 Transputer Memory Map

The left side of figure 5-2 shows the default memory map for the transputer module with Express loaded. In addition to the 1 Megabyte of external RAM, the transputer chip has 4 KBytes of on-chip RAM. Since this RAM is on-chip with the processor it is extremely fast. Using compiler switches, the memory map can be modified as shown in the right side of figure 5-2. This modification allows the program's stack to make use of the fast on-chip RAM resulting in considerable improvement in program speed. The danger is that the stack will overwrite the low memory addresses and the system will crash catastrophically. Through trial and error the correct positioning of the stack can be found. This is described in the Express documentation.

The demo Mandelbrot program was re-compiled with the switch set to allow it to use the fast on-chip RAM for its stack. This resulted in a 20 percent improvement in speed.

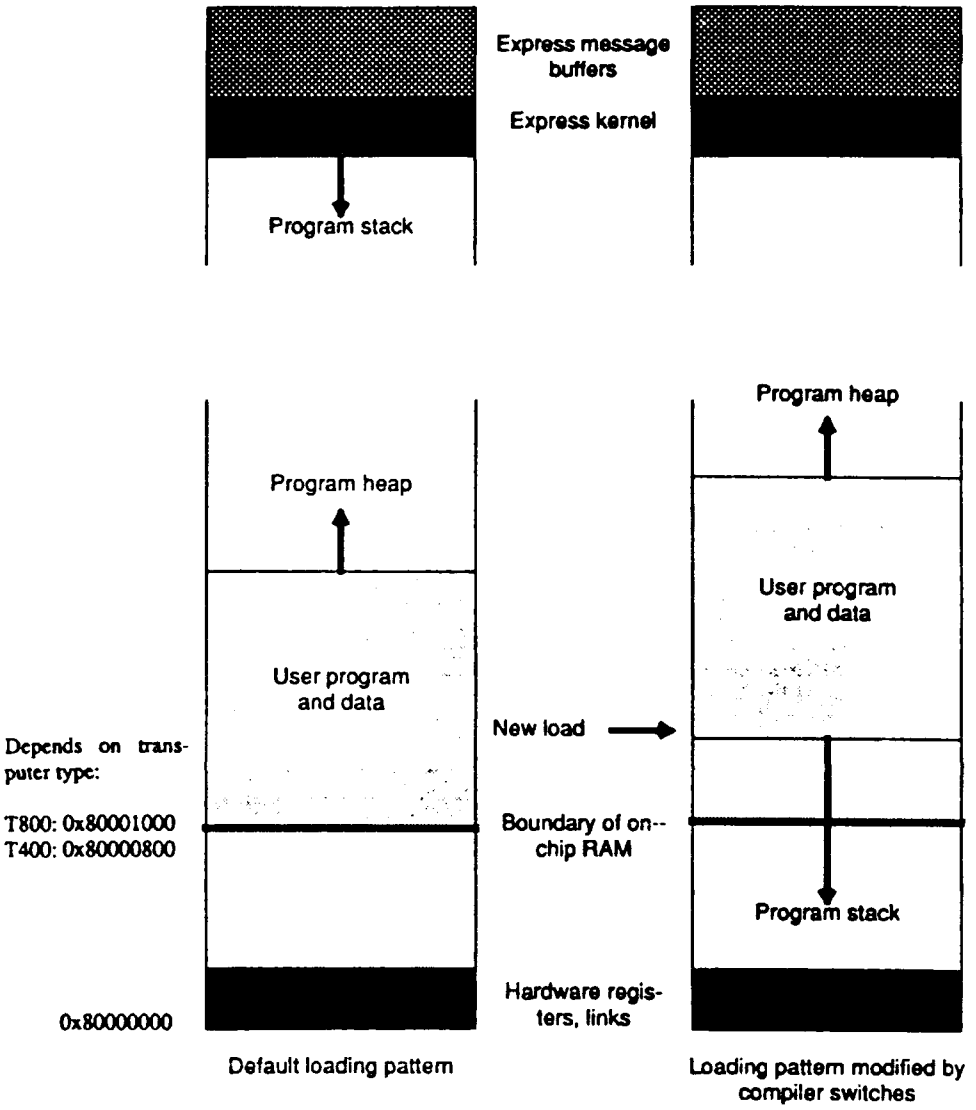


Figure 5-2. Transputer memory map with Express loaded.

5.3.3 VECLIB

ParaSoft sent an evaluation copy of the Sinectonalysis VECLIB library for vector and matrix operations on the transputer. The library contains over 250 routines to accelerate vector/matrix operations for the T800 series of transputers. It is hand-coded in T800 assembler language and offers a factor of 3-5 times the performance of C, Fortran, Pascal, or Occam. It is available for Occam, Logical Systems C, 3L C and Fortran running under a number of environments including Express.

We decided not to purchase the VECLIB for a number of reasons. One reason was that none of the 250 routines provided the precise functions needed. Also, since the functions were developed for C, modifications were required to implement them under Fortran. But the principal reason against purchasing was the desire to maintain code portability.

5.4 CURRENT ON THE TRANSPUTER

Once CURRENT was successfully compiled and run on the PC and the microVAX, the next step was to compile and run it on a single transputer. CURRENT compiled without any problems with the parallel compiler but did not give the correct answer when run. The problem was fixed by taking out the passing of function names as arguments in subroutines CGQ1 and SEGQAD. It is not clear why this was a problem. According to the parallel compiler documentation, function names can be passed as arguments.

The bus mouse was found to interfere with the transputer motherboard. It turned out that they both use IRQ line 3. The mouse was put on IRQ line 5, and the problem disappeared.

Several days were spent determining a suitable location in memory to load the Express kernel. The ParaSoft advice was to change the default location for Express loading to 0x800c8000. This turned out to be fine for running small programs, but when CURRENT was run on the transputer memory was exceeded. By trial and error it was found that CURRENT ran correctly if Express was loaded at location 0x800e0000.

With the sequential version of CURRENT running on a single transputer, the next step was to run it on four transputers simultaneously. This worked fine and showed that the time to run on four transputers was no different than running on one transputer. This indicates that there is little overhead to loading the program on more than one processor.

To make use of the 4K of fast on-chip RAM, the CURRENT program was loaded at 0x80000ff8. This put the stack in fast RAM. The achieved speedup was 20 percent.

A test was devised to determine the performance penalty for using Express as compared to a host-node program on the parallel compiler. A version of the demo Mandelbrot program (with no graphics) was implemented under Express. The result was that the Mandelbrot program ran as fast under Express as the same program did under the host-node 3L parallel compiler. Using the fast on-chip RAM for the stack gave a 20 percent speedup in operation under Express. The times for the Fortran Mandelbrot program with no graphics are give in table 5-2.

Table 5-2. Times for Fortran Mandelbrot program.

COMPUTER PLATFORM	TIME (s)
1 transputer running under Express:	25
1 transputer running under Express w/fast stack:	21
33 MHz 386 w/387 co-processor (Microsoft Fortran 5.0):	57
286 w/287 co-processor (Microsoft Fortran 5.0):	860
MicroVax:	100

6.0 PARALLEL MATRIX FACTOR AND SOLVE

A prime area in which parallel computing has been applied is the manipulation and solution of matrices. A large number of technical papers have been written in this area (Angus, et al., 1988; Fox, et al., 1988; Geist & Romine, 1988; and Heath & Romine, 1988). This chapter discusses the parallel matrix factor and solve routines that were implemented as part of this IED.

6.1 SEQUENTIAL LINPACK

As discussed in previous chapters, the JUNCTION formulation determines the currents on the wires, bodies, and junctions by solving the matrix equation shown in equation 3-22. This is done by first factoring the matrix by performing Gaussian elimination using LU decomposition with partial pivoting and then by solving for the current vector using backward and forward substitution. The library subroutines used are the LINPACK routines CGEFA and CGESL. CGEFA factors a general complex matrix. CGESL solves a complex matrix equation using the output from CGEFA.

LINPACK is a collection of Fortran subroutines that analyze and solve various systems of simultaneous linear algebraic equations (Dongarra, et al., 1979). The development of this package began in 1976. The LINPACK routines have been tested and validated on many different computers

6.1.1 CGEFA

A square matrix A can be factored into $A = LU$ where U is an upper triangular matrix and matrix L is the product of elementary lower triangular and permutation matrices. This factorization can be used to solve the linear equation $Ax = b$ by solving $L(Ux) = b$.

CGEFA uses Gaussian elimination with partial pivoting to compute the LU factorization of a complex matrix. The calling sequence is

CALL CGEFA(A,LDA,N,IPVT,INFO),

On entry,

- A — is a doubly subscripted complex array with dimension (LDA,N) which contains the matrix whose factorization is to be computed.
- LDA — is the leading dimension of the array A.
- N — is the order of the matrix A and number of elements in the vector IPVT.

On return,

- A — contains in its upper triangle an upper triangular matrix U and in its strict lower triangle the multipliers necessary to construct a matrix L so that A = LU.
- IPVT — is a singly subscripted integer array of dimension N which contains the pivot information necessary to construct the permutations in L. Specifically, IPVT(K) is the index of the K-th pivot row.
- INFO — is an integer which, if it is nonzero, technically indicates matrix singularity.

6.1.2 CGESL

CGESL uses the LU factorization of a complex matrix A (output from CGEFA) to solve linear systems of the form $Ax = b$ or $A^T x = b$ where A^T is the transpose of A. The calling sequence is

CALL CGESL(A,LDA,N,IPVT,B,JOB)

On entry,

A is a doubly subscripted complex array with dimension (LDA,N) which contains the factorization computed by CGEFA. It is not changed by CGESL.

LDA is the leading dimension of the array A.

N is the order of the matrix A and the number of elements in the vectors B and IPVT.

IPVT is a singly subscripted integer array of dimension N which contains the pivot information from CGEFA.

B is a singly subscripted complex array of dimension N which contains the right hand side b of a system of simultaneous linear equations $Ax = b$ or $A^T x = b$.

JOB indicates what is to be computed. If JOB is 0, the system $Ax = b$ is solved and if JOB is nonzero, the system $A^T x = b$ is solved.

On return,

B contains the solution vector, x.

6.2 PARALLEL LINPACK

When this IED project was initiated, general parallel versions of the LINPACK subroutines were not available. This is because parallel subroutines are by their nature very machine-specific and one of the goals of the LINPACK project was for the subroutines to be machine independent.

On sequential computers, the number of computations required to factor a matrix into a product of triangular matrices is of the order (n^3) where n is the dimension of the matrix. In comparison, the number of computations required for the triangular solution are of the order (n^2). Because of this, unless multiple solutions are required, most effort has focused on optimizing the factorization algorithm.

On parallel computers, good efficiency is more difficult to obtain for matrix solving compared to matrix factoring. This is because much more interprocessor communication is required during solving. Solving is inherently a finer grain process than factoring. Various parallel algorithms have been proposed. Each one has its advantages and disadvantages that are dependent on the number of processors being used, the size of the problem being solved, and the relative cost of communication and computation.

Good load balancing during matrix solving is achieved by using column-wrap mapping to distribute the matrix columns onto the processors. Column-wrap mapping involves dis-

tributing the columns sequentially onto the processors. This interleaves the columns onto the processors. Column-wrap mapping has good load balancing properties and gives excellent performance when doing LU decomposition, so this was the only scheme considered.

Ed Kushner at Intel Corporation (e-mail kushner@isc.intel.com) sent via e-mail the source code for Intel's parallel versions of DGEFA and DGESL from LINPACK. DGEFA and DGESL are double precision noncomplex versions of CGEFA and CGESL respectively. Included were two different matrix solving routines: a cyclic version and a wavefront version of DGESL. The differences between these versions are described below. These routines were all developed for Intel's iPSC parallel computer so a number of modifications were needed to implement them under Express on the transputer array.

6.2.1 Parallel Matrix Factor

The parallel version of DGEFA is called PGEFA. The matrix factoring routine is straightforward to parallelize and provides very high efficiencies. This routine assumes column-wrap mapping. The calling sequence is:

CALL PGEFA(A,LDA,N,M,P,ID,IPVT,BUF1,BUF2),

On entry,

- A — is a doubly subscripted array with dimension (LDA,N) which contains the matrix whose factorization is to be computed.
- LDA — is the leading dimension of the array A.
- N — is the order of the matrix A and number of elements in the vector IPVT.
- M — is the number of columns of A on each node (M = N/P).
- P — is the number of nodes (processors) allocated.
- ID — is the processor i.d. number.

On return,

- A — contains in its upper triangle an upper triangular matrix U and in its strict lower triangle the multipliers necessary to construct a matrix L so that A = LU.
- IPVT — is a singly subscripted integer array of dimension N which contains the pivot information necessary to construct the permutations in L. Specifically, IPVT(K) is the index of the K-th pivot row.

Notes:

BUF1 and BUF2 are work space buffers each of dimension N+1. They are used during communication between processors.

6.2.2 Parallel Matrix Solve

The problem being considered is the solution of the lower triangular linear system

$$Lx = b,$$

where L is a lower triangular matrix of order n, b is a known vector of dimension n, and x is the unknown solution vector of dimension n. The sequential version of LINPACK solves this by forward substitution using the following doubly nested loop:

for $j = 1$ to n
  $x_j = b_j / L_{jj}$
  for $i = j+1$ to n
    $b_i = b_i - x_j L_{ij}$.

There are a number of ways to parallelize this problem on a distributed memory parallel computer. Several algorithms have been developed. Two of the most popular are the wavefront algorithm and the cyclic algorithm (Heath & Romine, 1988). These algorithms assume the matrix is column-wrap mapped onto the processors.

The wavefront algorithm breaks the updating of b into segments and pipelines the segments through the processors in a wavefront fashion. The n-vector z is used to accumulate the updates of b so that the components of b remain distributed among the processors. The size of the circulated segments is an adjustable parameter that controls the granularity of the algorithm and therefore affects performance. The optimal value of the segment size depends on the characteristics of the hardware.

The wavefront algorithm is written in pseudocode as:

for $j \in$ *mycols*
  for $k = 1$ to # *segments*
    receive *segment*
    if $k = 1$ then
      $x_j = (b_j - z_j)/L_{jj}$
      segment = *segment* - $\{z_j\}$
    for $z_i \in$ *segment*
      $z_i = z_i + x_j L_{ij}$
    if |*segment*| > 0 then
      send *segment* to processor *map*(j + 1).

The cyclic algorithm is similar to the wavefront algorithm in that they both send a segment z between processors. However, the cyclic algorithm circulates a single segment of the fixed size p-1, where p is the number of processors. The name cyclic comes from the segment cycling through all other processors before returning to a given processor. The updates computed by the processor while the segment is circulating elsewhere are stored in the vector t. The cyclic algorithm is written in pseudocode as:

for $j \in$ *mycols*
  receive *segment*
  $x_j = (b_j - z_j - t_j)/L_{jj}$
  segment = *segment* - $\{z_j\}$
  for $z_i \in$ *segment*
    $z_i = z_i + t_i + x_j L_{ij}$
  $z_{j+p-1} = t_{j+p-1} + x_j L_{j+p-1,j}$

$segment = segment \cup \{z_{j+p-1}\}$
send *segment* to processor *map*(*j*+1)
for $i = j + p$ to n
  $t_i = t_i + x_j L_{ij}$.

Theoretical analysis shows that the cyclic algorithm performs best on a small number of processors whereas the wavefront algorithm performs best on a large number of processors. The performances of the two algorithms cross at about P=16, where P is the number of processors.

The cyclic parallel version of DGESL is called PGESL1. The wavefront parallel version of DGESL is called PGESL2. The calling sequence for both versions of PGESL is:

CALL PGESL(A,LDA,N,M,P,ID,IPVT,B,WORK,MSG)

On entry,

A is a doubly subscripted complex array with dimension (LDA,N) which contains the factorization computed by CGEFA. It is not changed by CGESL.

LDA is the leading dimension of the array A.

N is the order of the matrix A and the number of elements in the vectors B and IPVT.

M is number of columns of A on each node assuming column wrap mapping.

P is the number of nodes (processors) allocated.

ID is the processor i.d. number.

IPVT is a singly subscripted integer array of dimension N which contains the pivot information from CGEFA.

B is a singly subscripted complex array of dimension N which contains the right hand side b of a system of simultaneous linear equations $Ax = b$.

On return,

B contains the solution, x.

Notes:

WORK is a work space vector of dimension N

MSG is a work space vector of dimension P

In addition, the user might also want to pass the segment size to the wavefront algorithm routine.

6.2.3 Implementation of Parallel LINPACK under EXPRESS

A number of modifications were needed to the Intel parallel LINPACK routines to implement them on the transputer array operating under ParaSoft Express. These modifications fell into three categories:

1. converting double precision variables to complex variables,

2. using complex LINPACK routines instead of the equivalent double precision LINPACK routines, and
3. using Express communication functions/routines instead of iPSC communication functions/routines.

This last category was by far the most difficult to implement for reasons given below:

- Not being familiar with the iPSC made determining the effect of a given communication function/routine difficult.
- Sometimes it was difficult to determine the effect of the parameters being passed to a communication routine.
- Occasionally Express did not have an equivalent communication function-/routine.

All these difficulties were eventually overcome.

Converting from double precision to complex variables was a simple process of replacing the DOUBLE PRECISION type statements with COMPLEX type statements and making sure initialization was handled correctly.

Table 6-1 lists the names of the double precision LINPACK routines needed to be replaced with the listed complex counterparts.

Table 6-1. LINPACK routines.

LINPACK ROUTINES	
Double Precision	Complex
DAXPY	CAXPY
DCOPY	CCOPY
DFILL	CFILL
DSCAL	CSCAL
DSWAP	CSWAP
DVADD	CVADD
IDAMAX	ICAMAX

Table 6-2 lists the iPSC communication routines/functions and the equivalent Express communication routines/functions as well as a description of the operation each one performs.

Other modifications needed to be made included replacing the "include fcube.h" line in the iPSC program with the Express common block. Also, at the end of the matrix solving, the iPSC routines collected the answer in node 0. This was modified so that the answer was

Table 6-2. Communication routines/functions for the iPSC and Express.

COMMUNICATION ROUTINES/FUNCTIONS		
iPSC	Express	Description
Subroutine CSEND	Function KXWRIT	Send a message node to node
Function IRECV	Function KXRECV	Read a message (nonblocking)
Subroutine MSGWAIT(msg)	Parameter MSG from KXRECV	Check (MSG) until it is >= 0. (Can set up Fortran loop to do this)
IF (ID.EQ.x) CALL CSEND ELSE CALL CRECV ENDIF	Function KXBROD	Interprocessor broadcast
Subroutine CRECV	Function KXREAD	Read a message (blocking)
Subroutine GCOL	Function KXCONC	Collect and concatenate data in a set of nodes
Function INFONODE()	Parameter ISRC from KXREAD	Source node for message

collected/broadcast to all nodes under Express. The wavefront routine was modified so that the segment size (ISEG_SIZE) was passed as an input parameter in the argument list instead of being set to a value of 12.

6.2.4 Results

Performance measurements were made of both the parallel matrix factor routine and the two parallel matrix solve routines. In order to measure performance, timing routines were included in the algorithms using the Express timing function KXTIME. Timings were made using one, two, three and four processors. The timing measurements were converted to the standard performance measurement parameters, speed-up and efficiency, using equations 2-1 and 2-2.

Due to memory limitations, examples with more than 200 unknowns could not be run on a single transputer. Several sample problems were developed to test out the parallel factor and solve routines. The four sample problems had the following number of unknowns: 104, 116, 155, and 189.

6.2.4.1 Factor. Figure 6-1 shows the timing results on the parallel version of CGEFA for the four sample problems. Figure 6-2 shows speed-up and figure 6-3 shows efficiency. The important issue is that as the number of unknowns increases the parallel matrix factoring routine becomes more efficient. This is expected since small problems do not achieve good load balancing and cannot benefit greatly from parallel computation. An efficiency of over 90 percent is considered very good.

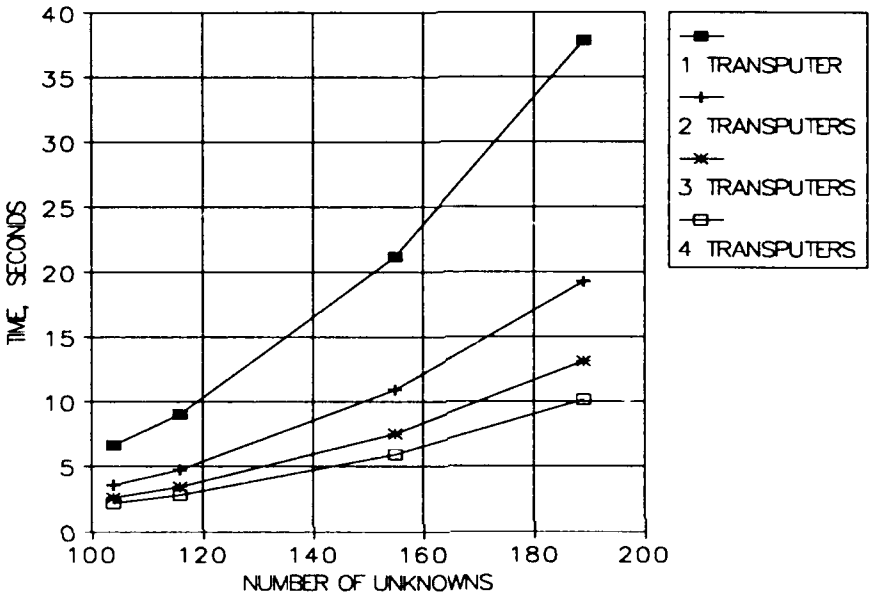


Figure 6-1. Timing for parallel matrix factoring routine.

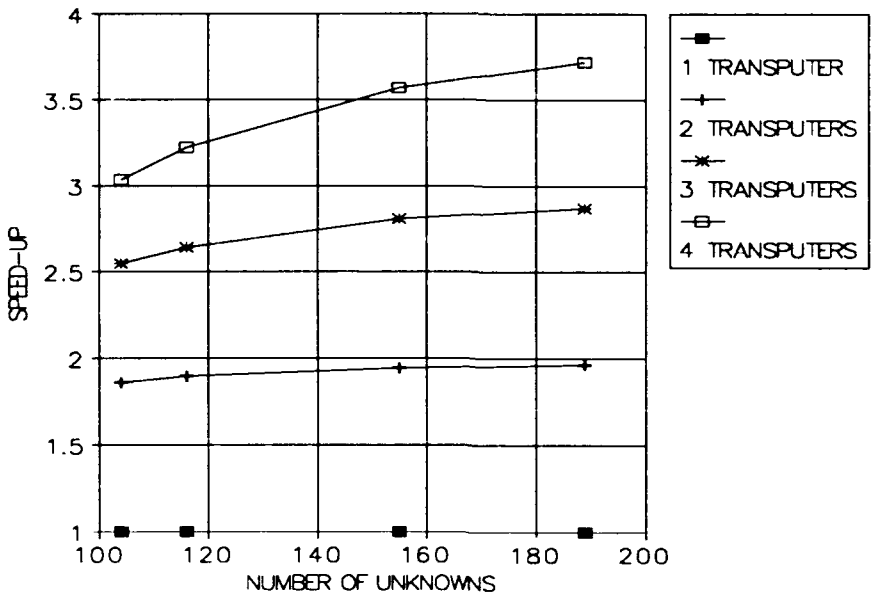


Figure 6-2. Speed-up achieved for parallel matrix factoring routine.

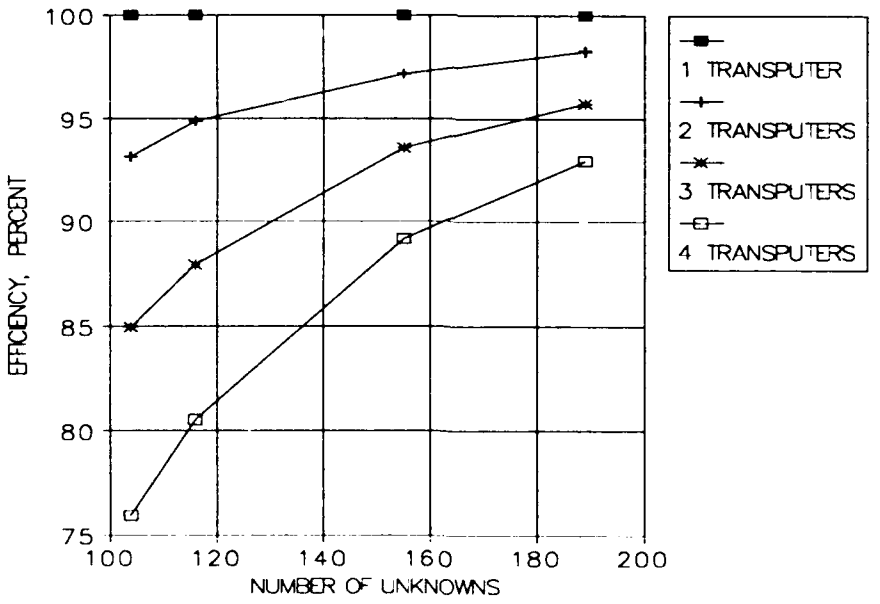


Figure 6-3. Efficiency achieved for parallel matrix factoring routine.

6.2.4.2 Solve. The performance of the two parallel matrix solve routines were measured for all the example problems. The results are shown here for the largest problem, the one with 189 unknowns. Table 6-3 shows the results for the cyclic algorithm.

Table 6-3. Performance results for cyclic algorithm for 189 unknowns.

Number of Processors	Time, seconds	Speedup	Efficiency (%)
1	0.621	1.00	100
2	0.367	1.69	84.5
3	0.257	2.42	80.7
4	0.207	3.00	75.0

The performance results for the wavefront algorithm are complicated by the presence of the adjustable segment-size parameter. This segment-size parameter can range from 1 to the number of unknowns. Table 6-4 shows the optimal performance segment size and time. Figures 6-4 through 6-7 show the timing results for matrix solve time using the wavefront algorithm as a function of segment size for one, two, three, and four processors. The optimal performance segment size appears to be a function of the number of processors (as well as the number of unknowns). In addition, by comparing the timing results for the wavefront algorithm with the results for the cyclic algorithm it can be seen that even using the optimum segment size the performance for the wavefront algorithm is much worse than the cyclic algorithm. Because of these two reasons the wavefront algorithm was rejected for use on the size of problem which could by run on the four processor transputer array. However, it must be noted that the performance results could turn out to be very different on a high performance computer with many more processors running larger problems.

Table 6-4. Performance results for wavefront algorithm for 189 unknowns.

Number of Processors	Optimal segment size	Time, seconds
1	95	0.741
2	72	0.555
3	34	0.448
4	24	0.380

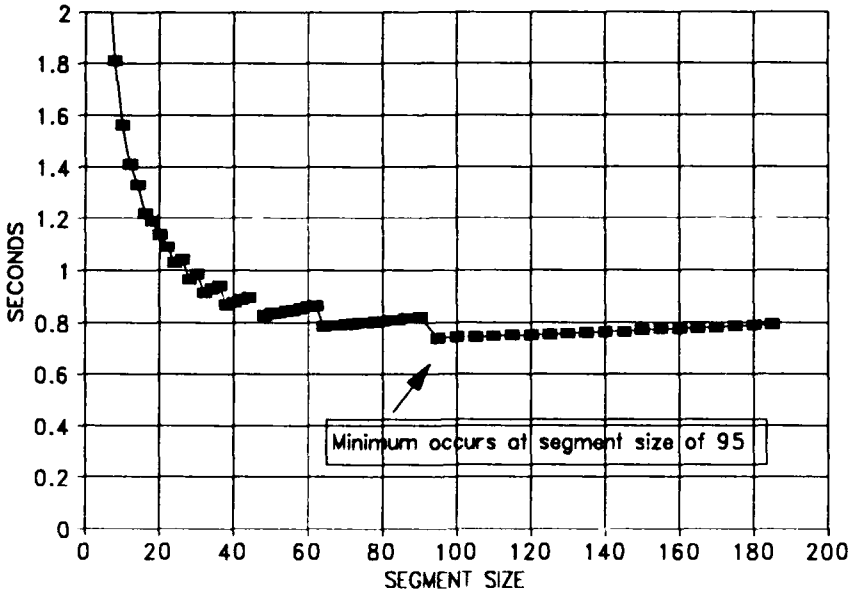


Figure 6-4. Timing for wavefront algorithm using 1 processor, 189 unknowns.

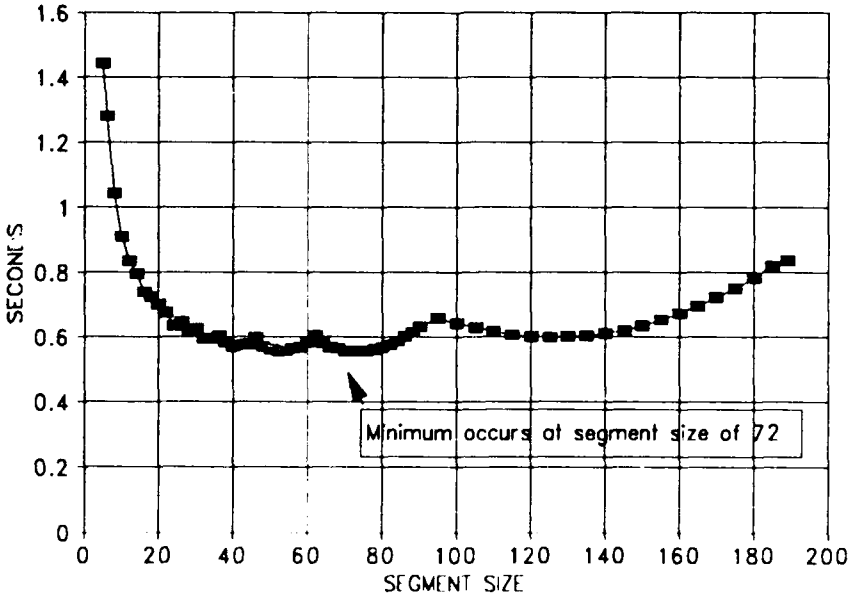


Figure 6-5. Timing for wavefront algorithm using 2 processors, 189 unknowns.

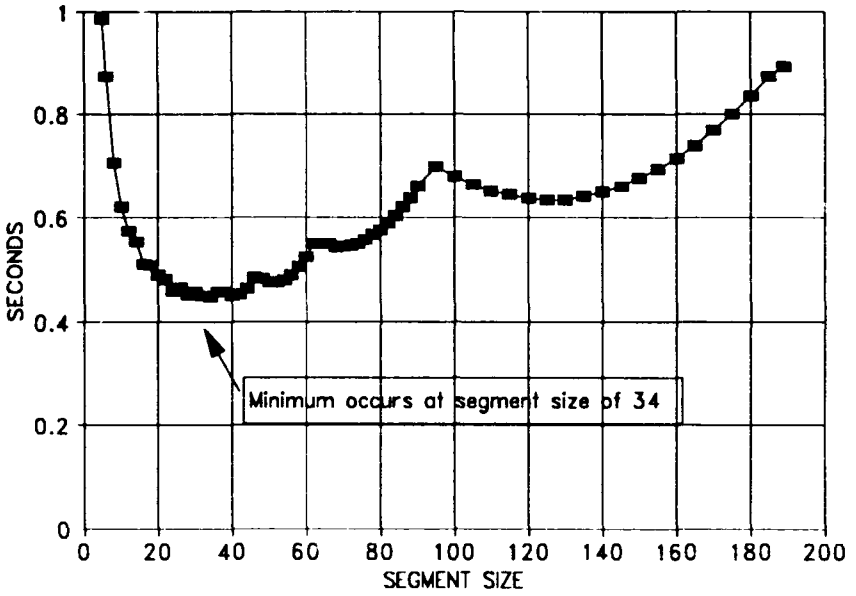


Figure 6-6. Timing for wavefront algorithm using 3 processors, 189 unknowns.

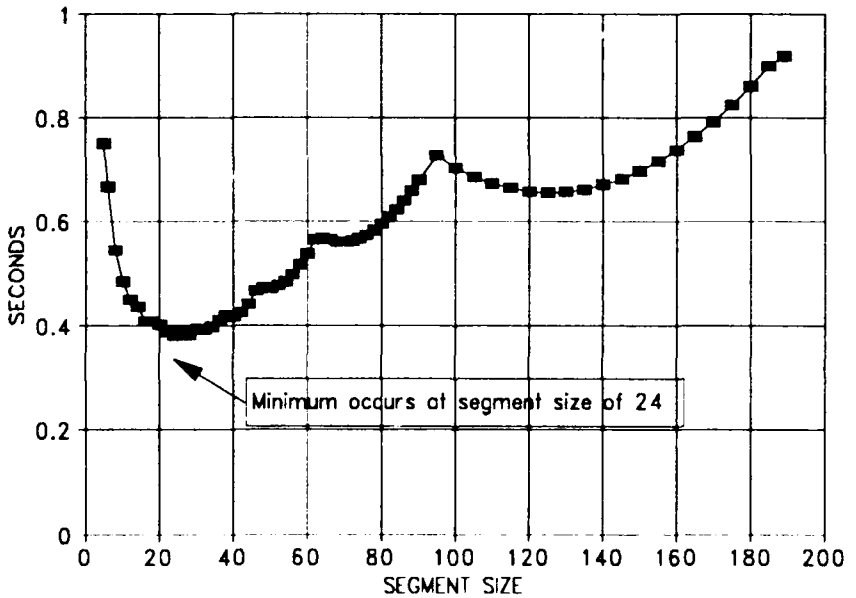


Figure 6-7. Timing for wavefront algorithm using 4 processors, 189 unknowns.

7.0 PARALLEL MATRIX FILLING

Developing a highly efficient general routine for parallel matrix filling was much more difficult in comparison to the work required to parallelize the matrix factor and solve routines. This was due to several reasons: the large variety of problem geometries being solved, shared calculations between matrix columns, memory limitations, and concurrent filling of the excitation vector.

7.1 SEQUENTIAL MATRIX FILLING

In JUNCTION the columns of the matrix correspond to source points on the structure. The rows of the matrix correspond to observation points on the structure. The body columns/rows come before the wire columns/rows, but otherwise the ordering of the columns/rows depends on how the problem was specified in the input data portion of the code (DATGN). The junction points are scattered within the wire portion of the matrix. Wire columns correspond to nodes on the wires whereas body columns correspond to sides of triangular patches on the surfaces. A junction is a node on a wire, hence its location is in the wire portion of the matrix. Table 7-1 shows the layout of the matrix.

Table 7-1. Layout of the impedance matrix.

	SOURCE POINTS	
OBSERVATION POINTS	Body Source Body Observation	Wire Source Body Observation
	Body Source Wire Observation	Wire Source Wire Observation

The filling of the matrix is performed in four subroutines: ZBB, ZWB, ZBW, and ZWW. The filling of the excitation vector is performed only in subroutines ZBB and ZWW. Table 7-2 shows in which subroutines the different portions of the matrix are filled.

Table 7-2. Subroutines used for matrix filling.

SOURCE POINT (COLUMN)	OBSERVATION POINT (ROW)	SUBROUTINE(S) USED
Body	Body	ZBB
Body	Wire	ZWB
Body	Junction	ZBB, ZWB
Wire	Body	ZBW
Wire	Wire	ZWW
Wire	Junction	ZWW, ZBW
Junction	Body	ZBB, ZBW
Junction	Wire	ZWW, ZWB
Junction	Junction	ZWW, ZBB

7.2 COLUMN-MAPPING TECHNIQUES

A wide variety of techniques can be used to map the columns of the matrix onto the processors. A column-wrap-mapping technique was used for matrix factoring and solving as described in the previous chapter.

It quickly became apparent that column-wrap mapping was not necessarily the best method to use for parallelizing the matrix-filling portion of the code. Adjacent columns of the matrix are often associated with each other. For bodies, adjacent columns are usually either on the same face or on an attached face. For wires, adjacent columns often refer to nodes joined by the same segment. In JUNCTION, calculations are made with respect to faces and segments, not edges and nodes. This avoids duplication of work in a serial computation. Alternative methods for mapping the columns onto the processors were needed. (Note: mapping the rows onto the processors (instead of the columns) was never considered since matrix factoring required the columns to be already mapped onto the processors).

Alternative mapping techniques were devised and evaluated. The techniques and performance results are described below in the section on structural dependencies. We found that using a different mapping technique to fill the matrix did not compromise the output from matrix solving. The only effect was to scramble the output vector. The output vector is easily and rapidly unscrambled at the end of the matrix solve routines. Unscrambling the output vector at the end is much faster than unscrambling the matrix before the matrix factor routine, since much less interprocessor communication is required for the former.

7.3 CODE MODIFICATIONS

Matrix filling was parallelized by modifying the loops within subroutines ZBB, ZWW, ZWB, and ZBW to require a given processor to do only the calculations associated with the matrix columns mapped onto the processor. This is fairly straightforward since with ParaSoft Express each processor knows its identification number as well as the total number of processors working on the problem.

Parallelizing matrix filling involved much more extensive modifications to the sequential version of the code than were required for parallelizing matrix factoring and solving. Not only did subroutines ZBB, ZWW, ZWB, and ZBW need to be changed, but initialization code was added at the beginning of CURRENT and modifications were made to the matrix solve routines to unscramble the output vector.

To determine which type of column mapping technique was desired another input file, FOR017, was added to CURRENT. This file contains just one value, an integer that is read into variable NWRAP. Table 7-3 shows the correspondence between the value read into NWRAP and the mapping techniques. These techniques are described in section 7.4 below.

Table 7-3. Correspondence between NWRAP and mapping technique.

NWRAP	COLUMN MAPPING TECHNIQUE
0	Pure block mapping
1	Wrap mapping
>=2	Block wrap mapping using a block size of NWRAP
-1	Random mapping
-2	1st order mapping

7.4 STRUCTURAL DEPENDENCIES

The performance of the parallel matrix-filling algorithm was found to be very dependent on the structure being analyzed. The first types of problems analyzed were for homogeneous structures. Homogeneous structures are defined as having either only body elements or only wire elements. Next, heterogeneous structures were evaluated. Heterogeneous structures are defined as having a mixture of body, wire, and junction elements.

7.4.1 Homogeneous Structures

Six types of structures were considered: straight wires, cylinders, plates, cones, disks, or spheres. The performance of the parallel matrix-filling routines was evaluated for homogeneous problems made up of each of these types of structures. Two column-mapping techniques were compared for each structure type. The two techniques used were column-wrap mapping and column-block mapping. Column-wrap mapping consists of interleaving the columns onto the processors. It is described in detail in section 7.2. Column-block mapping

consists of distributing the columns onto the processors using contiguous blocks of columns rather than individual columns. As an example, if 4 processors were being used to fill a matrix with 100 columns, using column-block mapping would mean that the first processor would fill the first 25 columns, the second processor would fill the second 25 columns, and so on.

Four sample problems were developed for each structure type. The number of unknowns ranged from 40 to 200. Each sample problem was run on both a single transputer and four transputers. Measurements were made of both matrix-filling time and matrix-solving time for both column-wrap mapping and column-block mapping.

Table 7-4 shows the measured times and calculated efficiencies for matrix filling for the different structures using wrap and block mapping. The data are displayed graphically in figures 7-1 through 7-3.

Table 7-4. Matrix-filling times/efficiencies for various structures.

Structure Type	Number of Unknowns	1 Processor	4 Processors			
			Wrap Mapping		Block Mapping	
		TIME,sec	TIME	EFFIC	TIME	EFFIC
WIRE	47	7.38	3.32	55.6%	1.97	93.7%
	97	30.78	14.06	54.7%	8.08	95.2%
	147	70.51	31.54	55.9%	17.95	98.2%
	197	126.06	56.91	55.4%	32.22	97.8%
CYLINDER	56	8.56	4.97	43.1%	2.80	76.4%
	104	27.04	15.78	42.8%	7.94	85.1%
	152	55.95	32.69	42.8%	15.68	89.2%
	200	95.17	55.62	42.8%	25.99	91.5%
PLATE	40	5.12	2.66	48.1%	1.66	77.1%
	96	25.11	12.69	49.5%	7.88	79.7%
	133	45.7	25.95	44.0%	14.01	81.5%
	176	76.87	39.15	49.1%	21.92	87.7%
CONE	52	7.15	4.08	43.8%	2.54	70.4%
	100	24.09	13.88	43.4%	7.36	81.8%
	155	55.85	29.12	47.9%	16.56	84.3%
	200	90.52	46.58	48.6%	25.31	89.4%
DISK	49	6.60	3.77	43.8%	2.77	59.6%
	98	25.19	13.7	46.0%	10.42	60.4%
	150	55.65	29.84	46.6%	20.05	69.4%
	200	97.01	53.12	45.7%	34.33	70.6%
SPHERE	60	9.06	4.82	47.0%	3.23	70.1%
	90	19.75	10.65	46.4%	6.85	72.1%
	168	66.56	37.98	43.8%	20.27	82.1%
	198	91.84	47.99	47.8%	28.96	79.3%

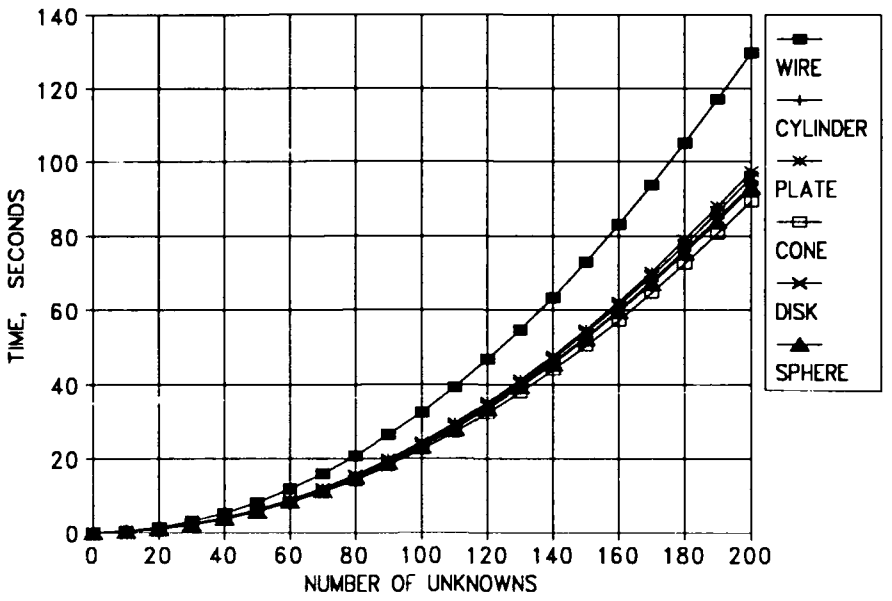


Figure 7-1. Matrix-filling time for one processor.

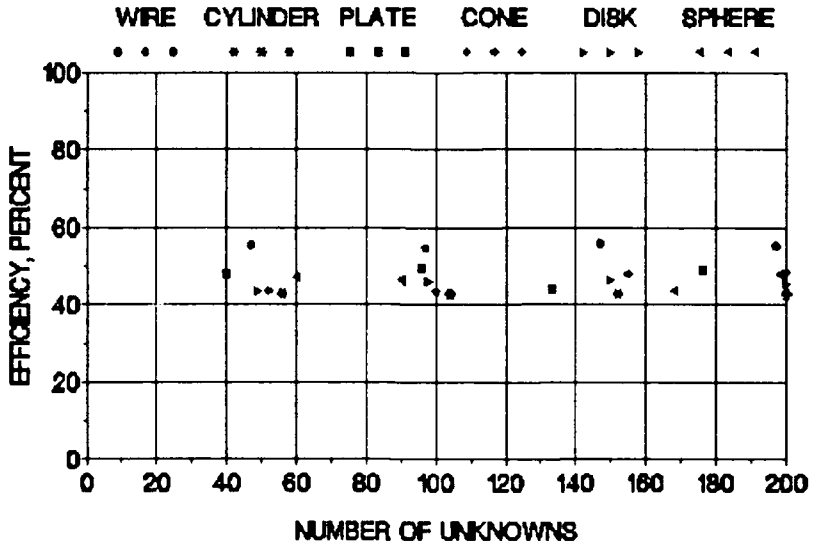


Figure 7-2. Matrix-filling efficiency for four processors, wrap mapping.

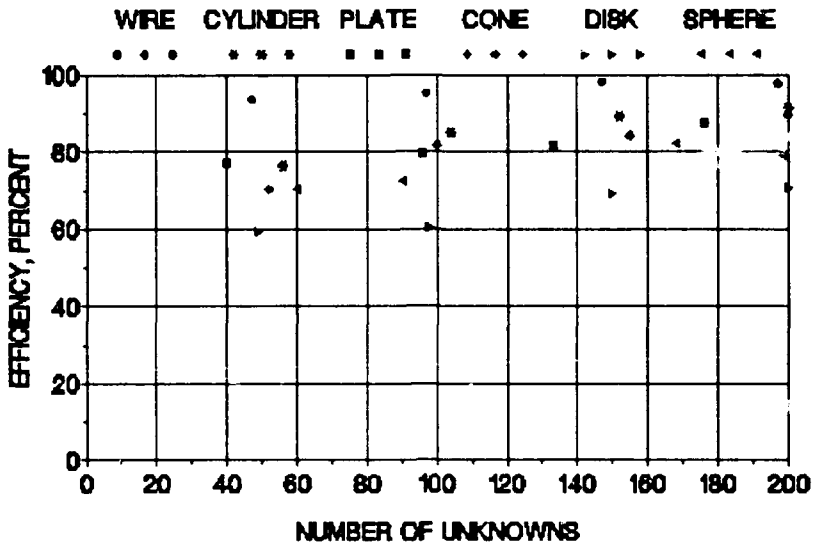


Figure 7-3. Matrix-filling efficiency for four processors, block mapping.

From the previous table and figures, several interesting observations can be made. For a single processor it is seen that an all-wire homogeneous matrix takes longer to fill than an all-body homogeneous matrix of the same size. On average, an all-wire matrix took about 38 percent longer to fill than the same size all-body matrix. The cylinder, plate, cone, disk, and sphere matrices all took about the same time to fill a given size matrix.

When the sample problems were run on four processors, distinct structural dependencies emerged for filling times and efficiencies. It is important to note that column-block mapping always performed better than column-wrap mapping. Column-block mapping filling efficiencies were almost always greater than 70 percent, whereas column-wrap mapping filling efficiencies were always between 40 and 60 percent. Interestingly, each structure performed differently in filling efficiencies. For column-wrap mapping, wires performed best, followed by plates, cones, spheres, disks, and, then, cylinders. For column-block mapping, wires again performed best, followed by cylinders, plates, cones, spheres, and, then, disks. These results are due to the nature of the physical connectivity of the various structures and how their elements are distributed to the columns of the matrix. Again, in JUNCTION calculations are made with respect to faces and segments. To avoid duplication of computation on the processors, it is advantageous to have all the edge computations for a given face to be on the same processor. For a wire, both nodes of a given segment should be on the same processor.

Table 7-5 shows the measured times and calculated efficiency for matrix factoring and solving for the different structures. The efficiency data are displayed graphically in figure 7-4.

Table 7-5. Matrix factor and solve times/efficiency for various structures.

Structure Type	Number of Unknowns	1 Processor	4 Processors	
		TIME (sec)	TIME (sec)	EFFIC (%)
WIRE	47	0.66	0.22	75.0
	97	5.34	1.48	90.2
	147	18.09	4.81	94.0
	197	42.94	11.19	95.9
CYLINDER	56	1.13	0.35	80.7
	104	6.70	1.81	92.5
	152	20.22	5.34	94.7
	200	45.36	11.79	96.2
PLATE	40	0.42	0.15	70.0
	96	5.12	1.44	88.9
	133	13.35	3.58	93.2
	176	30.55	8.04	95.0
CONE	52	0.92	0.29	79.3
	100	5.97	1.63	91.6
	155	21.45	5.63	95.2
	200	45.41	11.79	96.3
DISK	49	0.78	0.25	78.0
	98	5.60	1.54	90.9
	150	19.48	5.11	95.3
	200	45.32	11.78	96.2
SPHERE	60	1.37	0.41	83.5
	90	4.38	1.21	90.5
	168	27.10	7.11	95.3
	198	44.02	11.41	96.5

In matrix factoring and solving there are no structural dependencies. Note that as the number of unknowns increases, the factoring and solving efficiencies increase. For 200 unknowns factoring and solving efficiency is greater than 95 percent. Matrix-filling efficiency also increases as the number of unknowns increases.

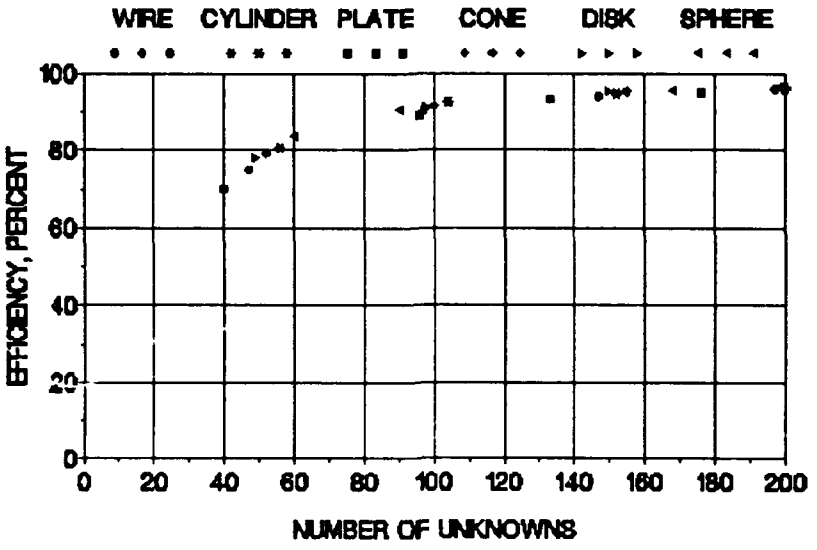


Figure 7-4. Matrix factor and solve efficiencies for 4 processors.

7.4.2 Heterogeneous Structures

The heterogeneous structures that were evaluated consisted mostly of square plates with numerous attached wires. This allowed the evaluation of matrices with various numbers of body, wire, and junction columns.

After evaluating a large number of heterogeneous problems, a number of observations were made. These observations are summarized below:

- The junction columns are scattered within the wire portion of the matrix. The distribution of the junction columns depends on how the problem is initially specified in DATGN. Fortunately, it is fairly straightforward to convert from junction number to matrix column number.
- A junction can connect to between one and six faces on a body. A quirk of JUNCTION is that the code will only recognize a maximum of one junction attached to a given face.
- To achieve maximum efficiency, the filling of a junction column should be done by the same processor filling the columns associated with the faces to which the junction is attached, since a large part of the calculations are in common. The problem here is that the associated body columns are sometimes scattered all through the matrix.
- The time to fill a matrix is a function of many things including: relative/absolute number of wire, body, and junction unknowns; number of separate wires; number of segments on each wire; locations of wires and junctions relative to a body; number of faces connected to each junction.

- Developing a general method for balancing the work load and achieving optimum efficiency for a heterogeneous problem is very complicated, even for the simple problems studied involving a single plate with various attached wires. If the choice is between block and wrap mapping, block mapping gives better results. (For the problems studied, block mapping was 62 to 93 percent efficient, whereas wrap mapping was 41 to 52 percent efficient).

Based on the above observations, a decision was made to implement two additional column-mapping techniques. Both techniques are modifications of the standard block-mapping technique.

The first technique is called random mapping. Random mapping is block mapping with one twist: instead of the junction columns staying grouped with the wire columns, the junction columns are filled by the processors which are filling the body columns of the matrix. The columns are redistributed to keep a balanced number of columns on each processor. As an example, suppose a matrix has 50 body columns, 50 wire columns, and 6 junctions, and the problem is being run on 4 processors. Under standard column-block mapping the first 25 body columns would be filled by the first processor, the second 25 body columns would be filled by the second processor, the first 25 wire columns would be filled by the third processor, and the second 25 wire columns would be filled by the fourth processor. The junction columns, being wire columns also, would be distributed on the third and fourth processors. Under random mapping the six junction columns would first be distributed to the first two processors in the following manner: junction 1 column to first processor, junction 2 column to second processor, junction 3 column to first processor, junction 4 column to second processor, and so on. The first and second processors would each have 3 junction columns. Block processing would then be used to distribute the body and remaining wire columns, so that each processor would finish up with 25 total columns as before.

The second technique is called first-order mapping. This is similar to the random mapping described above except now the junction columns are assigned to the body processors using knowledge of which processor will be calculating the majority of columns associated with the faces to which that junction is attached. This could potentially cause more junction columns to end up on one of the body processors than another, but the improvement in performance could be significant.

An analysis of matrix-filling efficiency was run on four processors for the problem of a square plate with 8 wires attached to it. The problem involved 96 body unknowns, 96 wire unknowns, and 8 junctions. The four different column-mapping techniques were used. The results are shown in table 7-6.

The results show that random mapping only gives a slight improvement over block mapping, but first-order mapping gives a significant improvement over block mapping.

Table 7-6. Efficiencies of various mapping techniques.

Mapping Technique	Efficiency (%)
WRAP	40.5
BLOCK	61.8
RANDOM	63.9
FIRST ORDER	76.9

8.0 RESULTS AND CONCLUSIONS

8.1 PERFORMANCE COMPARISONS

Comparisons were run for the microVax, a single transputer, the four- transputer array, and the Convex Model C-220. The Convex is a mini-supercomputer that can do some automatic vectorization.

Sample data sets were generated for both bodies and wires. Eight body data sets were generated with 104, 152, 200, 248, 296, 356, 404, and 452 unknowns. Eight wire data sets were generated with 47, 97, 147, 197, 247, 297, 347, and 397 unknowns. For each data set and each hardware platform timing measurements were made for both matrix filling and matrix factoring and solving. On the four-transputer array, the data sets were run using both column-wrap mapping and column-block mapping. On the Convex, measurements were made of both CPU time and actual elapsed time. At the time the measurements were made, there were 13 other users on the Convex and elapsed time averaged about 3.5 times longer than CPU time. This should be kept in mind when making comparisons between the various hardware platforms. Results are shown in tables 8-1 through 8-4.

Table 8-1. Time to fill all-body matrix, seconds.

Number of Unknowns	COMPUTER PLATFORM					
	MicroVax	1 Xputer	4 Xputers		CONVEX	
			Wrap Map	Block Map	CPU	Elapsed
104	73.7	26.6	16.7	8.8	5.1	20
152	151.7	55.0	33.4	16.5	10.7	26
200	257.5	93.8	56.1	26.6	18.3	69
248	394.0		85.4	39.7	27.9	105
296	555.9		120.1	54.9	39.4	145
356	801.6		172.8	78.0	56.9	201
404	1027.8		221.4	99.1	73.1	260
452	1282.3		276.1	122.9	91.4	329

Table 8-2. Time to factor and solve all-body matrix, seconds.

Number of Unknowns	COMPUTER PLATFORM					
	MicroVax	1 Xputer	4 Xputers		CONVEX	
			Wrap Map	Block Map	CPU	Elapsed
104	15.1	6.7	1.8	1.8	0.5	2
152	45.5	20.2	5.3	5.3	1.3	4
200	101.8	42.7	11.7	11.8	2.8	7
248	191.9		21.9	22.1	5.1	32
296	323.7		36.9	37.1	8.5	30
356	559.9		63.6	63.9	14.5	55
404	814.3		92.5	92.8	21.0	79
452	1137.2		129.0	129.4	29.1	102

Table 8-3. Time to fill all-wire matrix, seconds.

Number of Unknowns	COMPUTER PLATFORM					
	MicroVax	1 Xputer	4 Xputers		CONVEX	
			Wrap Map	Block Map	CPU	Elapsed
47	19.1	7.3	3.3	2.0	0.9	1
97	78.9	30.6	14.0	8.0	3.6	9
147	180.5	70.1	31.4	17.9	8.3	23
197	322.4	125.4	56.7	32.0	14.9	54
247	505.3		88.0	49.6	23.4	112
297	729.3		127.8	71.9	33.8	154
347	993.7		173.1	97.2	46.0	166
397	1302.1		227.8	127.8	60.2	172

The above data were curve fit to equations. Matrix filling should be proportional to N^2. Matrix factoring and solving should be proportional to N^3. (Note: Matrix solving should be proportional to N^2, but since factoring takes much longer than solving, factoring plus solving is effectively proportional to N^3.)

Table 8-4. Time to factor and solve all-wire matrix, seconds.

Number of Unknowns	COMPUTER PLATFORM					
	MicroVax	1 Xputer	4 Xputers		CONVEX	
			Wrap Map	Block Map	CPU	Elapsed
47	1.6	0.7	0.2	0.2	0.1	0
97	12.3	5.3	1.5	1.5	0.4	1
147	41.2	18.1	4.8	4.8	1.2	5
197	97.4	42.9	11.1	11.1	2.7	10
247	189.9		21.5	21.5	5.0	26
297	327.8		37.0	37.0	8.6	35
347	520.2		58.6	58.6	13.4	39
397	776.0		87.4	87.4	19.8	57

The equation for filling time, T_{fill}, is

$$T_{fill} = aN^2.$$

The equation for factor and solve time, $T_{f\&s}$, is

$$T_{f\&s} = bN^3.$$

The proportionality constants, a and b, were calculated for each computer platform for both bodies and wires. The values are shown in table 8-5.

The results show that the microVax fills a wire matrix 32 percent slower than it fills a body matrix of the same size. Using column-wrap mapping the four transputers fill a wire matrix 8 percent slower than they do a body matrix. In contrast, the Convex fills a wire matrix 15 percent faster than it does a body matrix! This may be due to the Convex's automatic vectorizing abilities. For comparison, a single transputer fills a wire matrix 36 percent slower than it fills a body matrix. The 8 percent figure for the four transputers can be accounted for by the fact that the wire-filling efficiency (55 percent) is about 28 percent greater than the body-filling efficiency (40 percent) for these problems and 28+8=36. The conclusion here is that for the JUNCTION code, sequentially speaking, wire filling takes about one-third longer than body filling.

Some of the comparison data are displayed graphically in figures 8-1 through 8-3. The figures show comparisons between the four-transputer array, the Convex (CPU time), and projected results for a 16-transputer array. Results are projected for greater than 450 unknowns using the a and b proportionality constants from table 8-5. A four-transputer array with 1 Megabyte of RAM per node can solve problems with up to 450 unknowns. With a 16-transputer array, problems with more than 900 unknowns could be solved.

Table 8-5. Proportionality constants for bodies and wires.

COMPUTER PLATFORM	BODIES		WIRES	
	a	b	a	b
MicroVax	0.006247	1.23E-5	0.008248	1.24E-5
1 Transputer	0.00239	5.68E-6	0.00326	5.72E-6
4 Transputers - Wrap Mapping	0.001341	1.39E-6	0.001441	1.39E-6
4 Transputers - Block Mapping	0.000589	1.40E-6	0.000807	1.39E-6
Convex (CPU time)	0.000446	3.13E-7	0.000381	3.16E-7

The plots show that a 16-transputer array would perform better than the Convex for filling the matrix and almost as good as the Convex for factoring and solving the matrix. A 16-transputer array, including the motherboard, would cost about $15K.

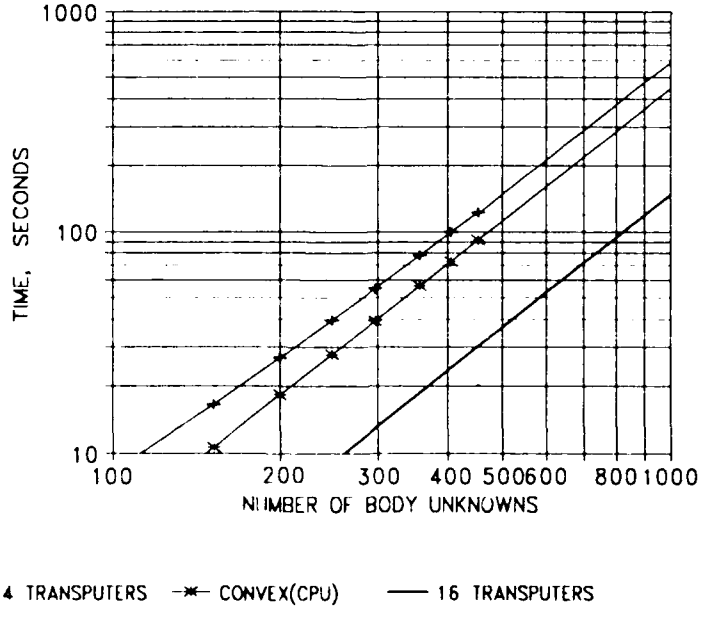


Figure 8-1. Matrix-filling time comparisons, block mapping.

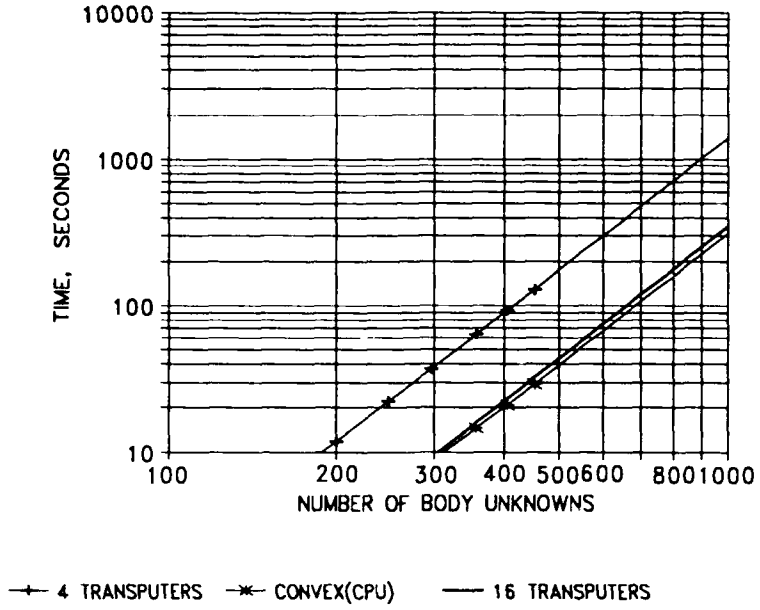


Figure 8-2. Matrix factoring and solving time comparisons.

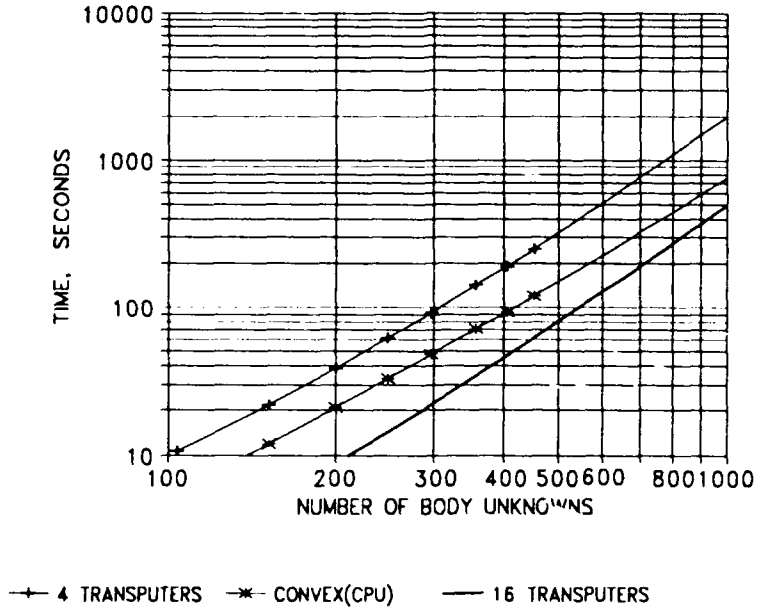


Figure 8-3. Total computation time comparisons, block mapping.

8.2 OBSERVATIONS

In the course of working on this IED project a number of observations were made regarding transputers, ParaSoft Express, and parallel processing in general:

- ***A transputer array is an inexpensive, flexible parallel processing platform.*** An array of four transputer modules, each with 1 Megabyte of RAM, plus the PC motherboard cost less than $5K. The transputer array is very flexible since the number of processors can be easily expanded (just plug more modules in the sockets) and any Fortran code which runs on the PC can be modified to run on the transputer array.
- ***ParaSoft Express has been of great utility.*** Without Express it would have been very difficult to make the progress that was made in parallelizing the code during the course of this IED investigation. In addition, major modifications to the existing code would have been required, if not a complete rewriting of the code. Express will allow the porting of the code to another computer platform to be much more direct.
- ***Running an existing program on one transputer is very straightforward using Express.*** By adding just a few lines of code, then recompiling using the parallel compiler, and linking the object files to Express, an executable module that runs on a single transputer can be created for virtually any Fortran program that runs on the PC.
- ***The effort required to parallelize a code is algorithm dependent.*** Some algorithms can be parallelized just by adding several lines of code. Other algorithms require extensive restructuring. Still other algorithms are not suited for implementation on a parallel computer and must be left as sequential code. It is not always obvious beforehand which algorithms will be efficient to parallelize and which won't.
- ***The optimal parallel code can be dependent on the number and type of processors as well as the size of the problem being solved.*** This was shown to be the case with the matrix solve algorithms. Two algorithms were tested: a cyclic implementation and a wavefront implementation. Although the cyclic implementation outperformed the wavefront implementation for our transputer array, the literature suggests that the wavefront implementation will be the best performer for a massively parallel computing platform solving larger problems.

8.3 PROJECT PAYOFFS

Several payoffs have resulted, or will result, from the work done on this IED project:

- ***A computer platform now exists for the efficient transition of computational electromagnetics to high-performance computing.*** The transputer array can be used to initiate parallelizing an existing computer code.
- ***This next generation increase in processing speed will permit more accurate modeling of ship topsides and extend the practical method of moments modeling of ships to high HF through low UHF bands.*** This will make new, innovative ship designs much more feasible.

- ***Since topside synthesis is accomplished by iterative analysis, design quality will significantly improve with the enhanced speed offered by high-performance computing.***

9.0 THE FUTURE

9.1 OTHER HARDWARE

One of the goals of this project, which unfortunately was not accomplished, was to port the code over to a high-performance computer using Express. The target computer for this was NOSC Code 70's iWarp computer, but problems occurred due to its late arrival and the substitution of an older version of processing chip would not allow Fortran code to run. The newer model of iWarp processor should be installed in January 1992.

9.2 NEW TECHNOLOGY

The state-of-the-art technology in the area of parallel processing changes monthly both for hardware platforms and software tools. The transputers used for this project will soon be outdated, replaced by faster and more flexible processing units. New features are continuously being added to Express to make it more powerful and adaptable.

9.2.1 Future Hardware

The big news in the world of transputers is the development of the T9000 (Pountain, 1990). This next generation processor is being developed by the same company, INMOS, which did the original pioneering work on the development of the transputer. According to the manufacturer the key features of the T9000 are a high-performance pipelined superscalar processor and major support for multiprocessing applications. Peak performance will be more than 150 MIPS and 20 MFLOPS, representing a major advance in parallel computing and high-speed communications. The design goals for T9000 were to enhance the transputer's position as the premier multiprocessing microprocessor, and to establish a new standard in single processor performance, while maintaining compatability with existing transputer products. Of course, there will be some delay between the release of the T9000 and the development of compatible motherboards and software.

9.2.2 Future Software

A great deal of effort is going into developing programming software for parallel computing environments. This software includes parallel compilers, debuggers, performance analysis tools, visualization tools, dynamic load balancing tools, and automatic parallelization tools. Most of the presently existing software in these areas are very crude.

ParaSoft is in the process of improving Express. New tools being added include VTOOL, ASPAR, and EXDIST. VTOOL will allow memory access visualization. ASPAR provides automatic parallelization of sequential C programs. EXDIST will provide dynamic load balancing. ParaSoft's ultimate goal is to run Express within a heterogeneous parallel processing environment, known as a "meta" computer. A "meta" computer is made up of a number of architecturally different computers which are networked together and perform as a single entity.

9.3 COMPUTATIONAL CODES

The goal of this IED project was to put a single computational code into a parallel processing environment. The JUNCTION code, developed at the University of Houston, is the latest MoM algorithm. A number of other codes are used for CEM and would benefit from being put into the parallel processing environment.

The MoM is used for lower frequency analysis (below 30 MHz). It is based on the solution of integral equations. For higher frequency analysis (above 300 MHz), a code such as NEC-basic scattering code (NEC-BSC) is needed. NEC-BSC is based on asymptotic techniques formulated in terms of the geometrical theory of diffraction (GTD) or uniform theory of diffraction (UTD). Two other codes which are also based on UTD are the ray tracing and casting model (RTC) and the multiple obstacle gain reduction model (GMULT). RTC was developed by Rockwell International. GMULT was developed at Georgia Tech. RTC and GMULT are NAVSEA EM Engineering tools.

NEC-BSC is used for analysis of the Fresnel region fields and far-field patterns of antennas in the presence of complicated scattering structures. RTC computes microwave electromagnetic fields. GMULT computes microwave antenna patterns and can be used to predict the near-field electromagnetic environment. It is based on an analytical algorithm and uses a spherical angle function (SAF) representation of the electric field.

NEC-BSC, RTC, and GMULT could all benefit from being put into a parallel processing environment.

10.0 GLOSSARY

AI	associate investigator
AT	advanced technology
BSC	basic scattering code
CEM	computational electromagnetics
CPU	central processing unit
EFIE	electric field integral equation
FPU	floating point processing unit
GMULT	multiple obstacle gain reduction model
GTD	geometric theory of diffraction
HF	high frequency
HPC	high-performance computing
I/O	input/output
IED	independent exploratory development
LLNL	Lawrence Livermore National Laboratory
MFIE	magnetic field integral equation
MFLOPS	mega floating-point operations per second
MHZ	megahertz
MIMD	multiple instruction multiple data
MIPS	millions of instructions per second
MoM	method of moments
NEC	numerical electromagnetic code
NEEDS	numerical electromagnetic engineering design system
PC	personal computer
PI	principal investigator
RAM	random access memory
RTC	ray tracing and casting model
SAF	spherical angle function
SIMD	single instruction multiple data
SISD	single instruction single data
TRAM	transputer module
UHF	ultrahigh frequency
UTD	uniform theory of diffraction
VLSI	very large-scale integration
XT	extended technology

11.0 REFERENCES

Angus, I., G. Fox, J. Kim, and D. Walker. 1988. *Solving Problems on Concurrent Processors, Volume II*, Prentice Hall, NJ.

Davidson, D.B. 1990. "A Parallel Processing Tutorial," *IEEE Antennas and Propagation Society Magazine*, April.

Dongarra, J.J., C.B. Moler, J.R. Bunch, and G.W. Stewart. 1979. *LINPACK User's Guide*, SIAM, Philadelphia, PA.

Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. 1988. *Solving Problems on Concurrent Processors, Volume I*, Prentice Hall, NJ.

Geist, G.A., and C.H. Romine. 1988. "LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures," *SIAM Journal of Scientific and Statistical Computing*, Volume 9, Number 4, July.

Harrington, R.F. 1968. *Field Computation by Moment Methods*, The Macmillan Company, NY.

Heath, M.T., and C.H. Romine. 1988. "Parallel Solution of Triangular Systems on Distributed-Memory Multiprocessors," *SIAM Journal of Scientific and Statistical Computing*, Volume 9, Number 3, May.

Hwu, S.U., and D. R. Wilton. 1988. "Electromagnetic Scattering and Radiation by Arbitrary Comfigurations of Conducting Bodies and Wires." NOSC TD 1325 (August). Naval Ocean Systems Center, San Diego, CA.

INMOS. 1988. *The Transputer Databook*, Bath Press Ltd, Bath, England.

Li, S.T., J.C. Logan, and J.W. Rockway. 1988. "Ship EM Design Technology," *Naval Engineers Journal*, May.

ParaSoft Corporation. 1990. *Express Fortran Reference Guide, Version 3.0,* Pasadena, CA.

ParaSoft Corporation. 1990. *Express Fortran User's Guide, Version 3.0,* Pasadena, CA.

Pountain, D. 1990. "Virtual Channels: The Next Generation of Transputers," *BYTE Magazine*, April.

Tazelaar, J.M. 1988. "Parallel Processing," *BYTE Magazine*, November.

TRANSTECH Parallel Systems Corporation. 1990. *Transtech TMB08 - PC Transputer Motherboard User Guide*, Ithaca, NY.

Wilton, D.R., and S.U. Hwu. 1988. "Junction Code User's Manual." NOSC TD 1324 (August). Naval Ocean Systems Center, San Diego, CA.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1 AGENCY USE ONLY *(Leave blank)*	2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	October 1991	Final: Oct 90 — Sep 1991

4 TITLE AND SUBTITLE
PARALLEL PROCESSING APPLIED TO COMPUTATIONAL ELECTROMAGNETICS

5 FUNDING NUMBERS
PE: 0602936N
PROJ: RV36I21
SUBPROJ: 82-ZF08-01
ACC: DN300186

6 AUTHOR(S)
L. C. Russell and J. W. Rockway

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Ocean Systems Center
San Diego, CA 92152-5000

8 PERFORMING ORGANIZATION REPORT NUMBER
TR 1450

9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Ocean Systems Center
San Diego, CA 92152-5000

10 SPONSORING/MONITORING AGENCY REPORT NUMBER

11 SUPPLEMENTARY NOTES

12a DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b DISTRIBUTION CODE

13 ABSTRACT *(Maximum 200 words)*

This investigation applied parallel processing techniques to a computational electromagnetic code to see if high-performance computing can improve the utility and efficiency of techniques used in ship electromagnetic designs. Efficient parallel matrix filling, factoring, and solving routines were developed, implemented, and evaluated. Several techniques for mapping the matrix columns onto the processors were also implemented and evaluated.

This report lists several benefits that have resulted, or will result, from work done on this Independent Exploratory Development (IED) project.

14 SUBJECT TERMS
parallel processing
high-performance computing
transputer
computational electromagnetic code
method of moments

15 NUMBER OF PAGES

16 PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
L. C. Russell	(619) 553–6132	Code 824